

البرمجة بلغة C++

الجزء الأول

الطبعة الأولى العربية

2000

تأليف: جون ر. هيوبارد أستاذ الرياضيات وعلوم الكمبيوتر جامعة ريتشموند

- يغطى جميع أساسيات المنهج ويكمل أى منهج دراسى
- أفضل وسيلة لمساعدة الطالب لتجعله متميزا فى الاختبارات ويحصل على أعلى الدرجات
- يحتوى الكتاب على الكثير من المسائل المحلولة



الدار الدولية للإستثمارات الثقافية

مصر

ملخصات شوم

نظريـــات ومســائـــل

فى

البرمجة بلغة C++

(الجزء الأول)

جون ر. هيوبارد

أستاذ الرياضيات وعلوم الكمبيوتر جامعة ريتشموند

ترجمـــة

أ.د. محمد إبراهيم العدوى

أستاذ ورئيس قسم الاتصالات جامعة حلوان

د. محمد السيد أبو الوفا

مدرس بقسم الاتصالات جامعة حلوان

د. رفعت سالم أحمد

مدرس بقسم القوى الكهربية جامعة حلوان

الدار الدولية للاستثمارات الثقافية

مصر

حقوق النشر

Schaum's Outlines of Programming With C++

by

JOHN HUBBARD

الدار الدولية للاستثمارات الثقافية

8 إبراهيم العرابى - النزهة الجديدة - مصر الجديدة - القاهرة - ج.م.ع.

ص.ب: 5599 هليوبوليس غرب/ القاهرة - تليفون: 2957655/2972344 فاكس : 2957655 (00202)

رقم الايداع : 2000/3184

I.S.B.N: 977-282-080-3

تقديـــم

يغطي هذا الكتاب كل مفاهيم ANSI/ISO في لغة ++C القياسية. فهو يشتمل على أكثر من 200 مثال ومسألة محلولة. ويرى المؤلف أن أحسن طريقة لتعلم البرمجة هى الممارسة متبوعة بمجموعة من الأمثلة مع التوضيح الكامل لها .

ولغة ++C أنشئت بواسطة بارني ستروستروب Biarne Stroustrup في سنة 1980. اعتماداً على لغة C ولغة Simula فإن لغة ++C أصبحت أكثر اللغات شيوعاً في البرمجة الموجهة ذات الهدف.

إن معظم الأشخاص الذين يتعلمون لغة ++C لديهم بعض الخبرة السابقة في البرمجة ولكن هذا الكتاب يفترض عدم وجود هذه الخبرة. فهو يقدم لغة ++C كأول لغة برمجة . لذلك فإن الذين لديهم خبرة سابقة يمكن أن يحتاجوا فقط إلى أن يتصفحوا الأبواب القليلة الأولى. ويمكن للقراء أخذ نسخة من برامج الأمثلة والمسائل المحلولة فى هذا الكتاب من صفحة المؤلف على الإنترنت:

http: // www.richmond.edu/~hubbard/C++book

أتقدم بالشكر إلى كل أصدقائي الزملاء والطلاب وأساتذة ماكجروهيل الذين ساعدوني في المراجعة البارعة لهذا الكتاب ومنهم جون الينو John Aliano و أرثور بيدرمان Arthur Biderman وفرانسيس منتثنج بو Francis Minhthing Bui بيتي دالي Pete Dailey وكرس هانز Chris Hanes وجون ب. هوبارد John B. Hubbard وأرني سجرجونسون Arni Sigurjonsson وأندري سومرس Andrew Somers وماريين وكار Maureen Walker ونات وثيرس Nat Withers ونقر بالشكر مهاراتهم في تصحيح الأخطاء.

في النهاية أود أن أعبر عن تقديري لزوجتي وزميلتي أنتاهوبارد Anita Hubbard التي راجعت النسخة كاملة وعملت معظم المسائل بما فيها الكثير التي ساهمت فيها بنفسي. أنا مدين لها كثيراً.

جون ر. هوبارد

ريشموند فيرجينيا

المحتويـــات

الفصل الأول

1

مقدمة للبرمجة بلغة C++
Introduction to Programming in C++

البرنامج هو مجموعة متتابعة من الأوامر أو التعليمات يتم تنفيذها بواسطة الحاسب (computer). كل برنامج يكتب بلغة ما من لغات البرمجة . لغة C++ (وتنطق سي بلس بلس) تعتبر واحدة من أحدث وأقوى لغات البرمجة المتداولة فهي تمكن المبرمج (programmer) من كتابة البرامج الموجهة المركبة الأفضل كفاءة.

هذا الباب يقدم بعض السمات الرئيسية للغة C++ . يجب أن تقوم بترجمة compile وتنفيذ كل مثال في هذا الباب.

1.1 برامـــج مبســطة :

مثالنا الأول يوضح الأجزاء الرئيسية لبرنامج C++

مثال 1.1 برنامج ترحيب العالم

```
# include <iostream.h>
// This program prints "Hello , World."
{
    cout << "Hello , World. \n" ;
    return   0 ;
}
```

تعليمة التوجيه Include # التي في السطر الأول ضرورية لأي برنامج له خرج، فهي تشير إلى ملف خارجي يسمى iostream.h يحتوي على معلومات عن الهدف cout . لاحظ أن أقواس الزاوية > و < ليست جزءاً من إسم الملف ولكنها تستخدم لبيان أن هذا الملف من مكتبة لغة C++ القياسية.

السطر الثاني في البرنامج هو ملاحظة أو تعليق comment ويبدأ بالشرطتين المائلتين // .

الملاحظات أو التعليقات توضع في البرنامج لتزويد القراء بالشرح والتوضيح عن خطوات البرنامج . وهذه الملاحظات لا يهتم بها المترجم (compiler) ولا تؤثر في تنفيذ البرنامج.

السطر الثالث في البرنامج يحتوي على الدالة الرئيسية ( ) main . وهذه الدالة يجب أن تكون موجودة في كل برنامج ++C ، فهي تخبر المترجم من أين يبدأ تنفيذ البرنامج. القوسين ( ) التابعين لكلمة main يجب أن يكونا موجودين أيضاً.

يحتوي السطر الرابع والسابع على القوسين } و { فقط . يوجد بين هذين القوسين جسم الدالة ( ) main وهما لازمين لكل برنامج ++C.

السطر الخامس يحتوي على الجملة الآتية :

```
cout << "Hello , World.\n"
```

وهذه الجملة تعني إرسال الرسالة "Hello , World.\n" إلى الهدف cout وتنطق "سي أوت" (see out). هذا الهدف هو مجرى خرج قياسي standard output stream عادة يكون شاشة الحاسب. الإسم cout يرمز إلى «خرج وحدة تحكم طرفية» هذا الخرج سيظهر كالآتي :

```
Hello , World.
```

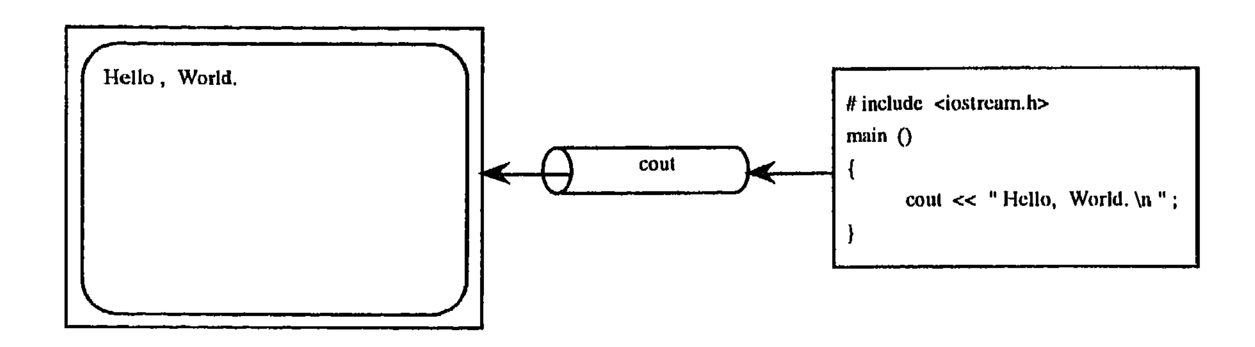
الرمز n\ هو رمز الإنتقال إلى سطر جديد. لاحظ أنه رمز واحد يتكون من الشرطة المائلة '\' وحرف 'n' . وجود هذا الرمز في نهاية الجملة بين علامتي إقتباس يخبر الحاسب بأن يبدأ سطر جديد بعد طباعة الحرف السابق للرمز n\ وهذا يعني نهاية السطر الحالي.

السطر السادس يحتوي على الجملة return 0. وهذه الجملة تنهي تنفيذ البرنامج والعودة إلى نظام التشغيل للحاسب . الرقم 0 يستخدم كإشارة بأن البرنامج إنتهى بنجاح. عبارة الخرج التي في السطر الخامس تحتوي على عديد من الرموز الشائعة في لغة سي ++. الرمز >> يسمى معامل الخرج (out put operator) أو معامل الإدماج (insertion operator) . فهو يدمج أو يضع الرسالة المراد إخراجها في مجرى الخرج (output stream) . وجود الرمز n\ في نهاية الرسالة يعني الانتقال إلى سطر جديد. وجود هذا الرمز في أي مكان في الرسالة المراد إخراجها يسبب نهاية السطر الحالي في الخرج وبداية سطر جديد . لاحظ أن كل من الرموز (>> و n\) يتكون من حرفين متجاورين ليس بينهما فرا غ أو مسافة.

لاحظ وجود الفاصلة المنقوطة في نهاية كل من السطر الخامس والسادس . كل جملة أو أمر في ++C يجب أن تنتهي بالفاصلة المنقوطة . ليس بالضروري أن تكون كل جملة في سطر مستقل . يمكننا وضع مجموعة

من الجمل على سطر واحد ويمكن وضع جملة واحدة على عدد من السطور. ليس المهم وضع الجملة على سطر واحد أو أكثر ولكن المهم أن كل جملة يجب أن تنتهي بالفاصلة المنقوطة.

نستطيع أن نتخيل علاقة الهدف cout للبرنامج وشاشة الحاسب كما في الشكل التالي:



مجرى الخرج cout يعمل عمل قناة تنقل الخرج من البرنامج إلى شاشة الحاسب (أو الطابعة أو أي جهاز خرج آخر) حرف بعد حرف.

البرنامج الذي في المثال 1.1 ليس هو أقصر برنامج ، بعض أجزاءه فقط هي المطلوبة لكل برنامج. في الحقيقة برنامج ++C لا يحتاج أن يحتوي على أي جمل (statements). بالطبع مثل هذا البرنامج يكون برنامج خالي أو فارغ "empty program" لا يقوم بتنفيذ أي شيء وهذا البرنامج الخالي هو أقصر برنامج ++C .

والمثال التالي يوضح أقصر برنامج ++C.

مثـال 2.1 أقصر برنامج ++C :

```
main ( ) { }
```

هـذا البـرنامـج الخـالـي أو الفـارغ لا يقوم بعمل أي شيء . هو ببسـاطة يبين الهـيكل المطلوب لكل برنامج ++C .

معظم برامج الترجمة (compilers) لا تحتاج للعبارة ;return 0 . بعض برامج الترجمة يعطي تحذير إذا حذفت هذه العبارة من البرنامج . كل مثال في الباب الأول سوف يحتوي على العبارة return 0.

نوصي أيضاً أن يحتوي كل برنامج في بدايته على ملاحظة أو تعليق يوضح ما يفعله البرنامج.

2.1 معامـل الخـرج

الرمز << يسمى معامل الإدماج insertion operator أو معامل الخرج output. فهو يدمج أو يضع خرج

البرنامج في مجرى الخرج المسمى على يساره. عادة نستخدم مجرى الخرج cout الذي يشير إلى شاشة الحاسب . لذلك عند تنفيذ الجملة cout <<66 سوف يظهر الرقم 66 على الشاشة. المعامل أحياناً يقوم بتنفيذ عملية واحدة أو أكثر . معامل الخرج » يقوم بإرسال قيمة الجملة التعبيرية الموجودة على يمينه إلى مجرى الخرج الموجود على يساره . وحيث أن تنفيذ هذا العمل يتم من اليمين إلى اليسار لذلك تم إختيار الرمز » ليمثل معامل الخرج. يجب أن تتذكر أن إتجاه السهم يشير إلى اليسار.

سبب تسمية cout بالمجرى (أو النهر) هو أن الخرج المرسل إليه يسير مثل المجرى أو النهر . لو أن مجموعة أشياء أرسلت إلى cout فإنها تسير في خط كما تسير نقط الماء في النهر واحدة بعد الأخرى. لذلك فإنها تظهر على الشاشة حسب ترتيبها.

مثــال 3.1

هذه النسخة من برنامجنا Hello World لها نفس الخرج كما في المثال 1.1

```
# include <iostream.h>
// This program illustrates the sequential output of three strings.
main ( )
{
      cout << "Hello , "<< "Wor"  <<  "ld.\n";
      return 0 ;
}
```

```
Hello, World.
```

في هذا المثال تم تجزئ الرسالة المراد إخراجها على الشاشة إلى ثلاثة أجزاء . حيث أن سطر الخرج ينفذ من اليسار إلى اليمين فإن كل جزء يرسل إلى مجرى الخرج حسب ترتيبه : الأول "Hello" ثم "wor" وأخيراً "ld.\n" حيث أنه لا يوجد أي رموز للإنتقال إلى سطر جديد بين الثلاثة أجزاء لذلك فإنهم يظهروا جميعاً في سطر واحد كما في المثال 1.1 . يستخدم مجرى الخرج cout عادة مع معامل الإدماج » في الصورة العامة الآتية :

cout << expressron << expression << ... << expression ;

تركيب هذه الجملة يبين أن cout يمكن أن يكون متبوعاً بزوج أو أكثر . كل زوج مكون من عامل الإدماج » متبوعاً بجملة تعبيرية . في المثال 3.1 يوجد ثلاثة أزواج .

3.1 الحروف characters وسلاسل الحروف string literals

كلمة "Hellow" تسمى سلسلة حروف string literal حيث أنها تتكون من مجموعة من الحروف بين علامتي إقتباس.

تعريف الحرف character هو أي حرف من حروف الهجاء أو رقم من الأرقام (0 إلى 9) أو علامة من العلامات مثل (+ و – و * و ٪ و ....) . معظم أجهزة الحاسب تستخدم نظام الشفرة الأمريكي القياسي لتبادل المعلومات (ASCII code) . أنظر الملحق A لمجموعة الشفرة الكاملة لهذا النظام . هذه المجموعة تحتوي على 52 حرفاً كبيراً upper case وحرفاً صغيراً lower case من حروف الهجاء وكذلك 10 أرقام من (0 إلى 9) وكل علامات الترقيم (– و + و * و ....) الموجودة على لوحة مفاتيح الحاسب وكذلك بعض الحروف التي لا تطبع . رمز الإنتقال إلى سطر جديد "\n" هو واحد من الرموز التي لا تطبع . هذا الرمز مكون من الشرطة المائلة \ وحرف n. يوجد أيضاً مجموعة أخرى من الرموز مكونة بهذه الطريقة مثل رمز ترك حقل خالي أفقي "\t" ورمز الإنذار الذي ينتج صفير عند الطباعة "\a" أيضاً لطباعة الشرطة المائلة نفسها يستخدم الرمز "\\" .

الحروف يمكن أن تستخدم في جملة البرنامج كجزء من سلسلة حروف أو كأهداف منفردة. عند الاستعمال المنفرد يجب أن تظهر في شكل ثوابت حرفية . الثابت الحرفي charecter constant هو حرف موجود بين علامتي إقتباس مفردة . ثوابت الحروف مثل الأهداف المفردة يمكن أن تخرج بنفس الطريقة التي تخرج بها سلسلة الحروف.

مثـــال 4.1 نسخة اخرى من برنامج الترحيب بالعالم :

هذه النسخة من برنامجنا Hello World لها نفس الخرج مثل النسخ السابقة من البرنامج .

```
# include <iostream.h>

// This program illustrates the ouput of strings and characters:

main ( )

{

    cout << "Hello , " <<  'W' << 'o' <<  'r' << "ld" << '.' << '\n';

    return 0;

}
```

```
Hello, World.
```

جملة الخرج في هذا البرنامج ترسل سبعة أهداف إلى cout : 2 سلسلة حروف هما "Hello" و "ld" و 5 ثوابت حرفية هي 'W' و 'o' و 'r' و '.' و '\n' .

بالطبع الحروف المفردة يمكن أن تستخدم في شكل سلسلة حروف . جملة خرج البرنامج السابقة يمكن أن تستبدل بالآتي :

```
cout << "Hello," << "W" << "o" << "r" << "ld" << "." << "\n";
```

هذه الجمل ترسل سبع سلاسل حروف إلى cout . لكن عند التعامل مع الحروف المفردة كأهداف مستقلة من الأفضل إستعمال ثوابت الحروف. سلاسل الحروف تخزن بطريقة مختلفة وتحتاج إلى حيز أكبر .

سلسلة الحروف التي لا تحتوي على أي حروف تسمى سلسلة حروف فارغة empty string ويرمز لها بالرمز " " . يمكننا طباعة رسالتنا باستخدام سلسلة الحروف الفارغة كالآتي :

```
cout << "Hello, Wo" << " " << "rl" << " " << " " << "d.\n" ;
```

4.1 طول سلسلة الحروف

طول سلسلة الحروف هو عدد الحروف التي تحتوي عليها . على سبيل المثال طول سلسلة الحروف "ABCDE" هو 5 حروف.

لغة ++C تحتوي على دالة خاصة تسمى () strlen التي يمكنك إستعمالها لمعرفة طول أي سلسلة حروف . وهذا موضح بالمثال التالي :

مثــــال 5.1

هذا البرنامج يطبع أطوال مجموعة من سلسلة الحروف :

```
# include <iostream.h>
# include <string.h>
// This program tests the strlen () function :
main ( )
{
    cout << strlen ("Hello , World. \n") << '\n' ;
    cout << strlen ("Hello , World. ") << '\n' ;
    cout << strlen ("Hello ,") << '\n' ;
    cout << strlen ("H") << '\n' ;
```

```
        cout << strlen (" ") << '\n' ;
        return 0;
}

14
13
7
1
0
```

الدالة ( ) strlen ببساطة تعد عدد الحروف الموجود في سلسلة الحروف المراد معرفة طولها. أول سطرين في خرج البرنامج هما العددان 14 و 13 وذلك يوضح أن رمز الإنتقال إلى سطر جديد n\ أعتبر كحرف واحد. سلسلة الحروف "Hello" طولها 7 حروف وسلسلة الحروف "H" طولها حرف واحد وسلسلة الحروف الفارغة " " طولها صفر.

الدالة ( ) strlen (وتنطق "ستيرلن") موجودة في ملف مستقل يسمى string.h يأتي مع بيئة البرمجة بلغة ++C . لذلك إذا كان برنامجك يحتاج لإستعمال الدالة ( ) strlen فيجب أن يحتوي على عبارة التوجيه include # التالية :

```
# include <string.h>
```

في مكان ما في البرنامج قبل كلمة ( ) main.

5.1 التعليقات Comments

يمكن أن يحتوي برنامجك على تعليقات لا يأخذها المترجم في الإعتبار . مثل هذه الرسائل يقرأها مستخدم البرنامج وتسمى تعليقات Comments.

يوجد نوعين من التعليقات في لغة ++C . تعليق لغة C القياسية وهو يبدأ بالرمز المركب من الشرطة المائلة مع النجمة */ وتنتهي بالرمز المركب من النجمة والشرطة المائلة /*.

أي شيء مكتوب بين هذين الرمزين سوف يهمل ولا يأخذه المترجم في الإعتبار . على سبيل المثال الجملة التالية تعتبر تعليق :

```
/* This is a C style Comment */
```

لغة ++C القياسية تبدأ بالشرطتين المائلتين // وتمتد إلى نهاية السطر. على سبيل المثال الجملة الآتية تعتبر تعليق :

```
// This is a C++ style Comment
```

معظم المبرمجين بلغة ++C يفضلون إستعمال الشرطتين المائلتين لأنها أسهل في كتابتها ورؤيتها في البرنامج. صيغة تعليق C ضرورية إذا كنت في حاجة إلى كتابة تعليق بداخل سطر في البرنامج ينفذ وهذا لا يفضل من الناحية العملية.

مثال 6.1 إستخدام نوعي التعليق :

هذا هو برنامجنا Hello World مع إضافة ستة تعليقات له :

```
/ ***************************************************************\
 *  Program to demonstrate comments                              *
 *  Written by J. R. Hubbard                                     *
 *  June 10, 1996                                                *
 *  Version 1.5                                                  *
\ ***************************************************************/
# include <iostream.h>            // this directive is needed to use cout
//     This prints message :   "Hello, World.":
main ( )
{
    cout << /* now printing */  "Hello , World.\n";  /* change */
    return 0;   // some compilers will complain if you omit this line
}  /* end of program*/
```

```
Hello, World.
```

هذا مثال لبرنامج مزود بالتعليقات ويوضح الإستخدامات الرئيسية لها. أول تعليق هو الستة سطور الأولى في أعلى البرنامج التي تعرف البرنامج والمبرمج . لاحظ أول حرفين (في بداية السطر الأول) عبارة عن شرطة مائلة ونجمة */ وآخر حرفين في السطر السادس عبارة عن نجمة وشرطة مائلة /* . التعليق الثاني في السطر السابع يبدأ بالشرطتين المائلتين // . وهي توضح الملاحظة داخل السطر وتكون على يمين الجملة المراد

إيضاحها أو شرحها. التعليق الثالث يشغل كل السطر الثامن. وهو يسبق كلمة ( ) main ويوضح بإختصار ما يفعله البرنامج . التعليق الرابع موجود بداخل عبارة الخرج. وهذا غير مرغوب فيه . التعليق الخامس موجود في نهاية السطر الذي يحتوي على عبارة الخرج . التعليق السادس موجود في نهاية البرنامج . المثال التالي يبين برنامجنا "Hello World" مع كيفية كتابة التعليقات بلغة ++C .

مثال 7.1 إستعمال الشرطتين المائلتين فقط في كتابة الملاحظات

هذه النسخة تبين كيف أن كل التعليقات الهامة تكتب بسهولة باستعمال الشرطتين المائلتين :

```
//-----------------------------------------------------------------------------------------------
//    Program to demonstrate comments
//    Written by H. R. Hubbard
//    June 11, 1996
//    Version 1.6
//-----------------------------------------------------------------------------------------------
# include  <iostream.h>                    // This directive is needed to use cout
//   Prints message:  "Hello, World.":
main ( )
{
     cout << "Hello, World.\n";   // change ?
     return 0;                 //   Some compilers will complain if you omit this line
}
```

لاحظ أن هذه التعليقات محدودة بالشرطتين المائلتين إلى آخر السطر فقط ولا يمكن أن تمتد إلى أكثر من سطر إلا إذا بدأ كل سطر بالشرطتين المائلتين .

6.1 المتغيرات Variables والأهداف Objects والإعلان عنها

المتغير هو رمز يمثل مكان للتخزين في ذاكرة الحاسب. المعلومة التي تخزن في هذا المكان تسمى قيمة المتغير value. أكثر الطرق شيوعاً لحصول المتغير على قيمة هي طريقة التخصيص assignment . وهذه الطريقة تأخذ الشكل التالي :

```
variable = expression;
```

التعبير expression تقدر قيمته أولاً ثم تخصص هذه القيمة للمتغير variable. علامة التساوي "=" تسمى معامل التخصيص assignment operator في لغة ++C.

مثـــال 8.1

هذا هو برنامج بسيط بلغة سي ++ يستعمل متغير للأعداد الصحيحة يسمى n:

```
# include <iostream.h>
// A simple example to illustrate assignment :
main ( )
{
      int   n;
      n = 66;
      cout  <<  n <<  endl;
      return   0;
}
```

```
66
```

السطر الأول بين القوسين { } يعلن عن أن n متغير من نوع الأعداد الصحيحة int . الجملة التي في السطر الثاني تخصص القيمة 66 للمتغير n .

لاحظ إستعمال الثابت الرمزي endl. إرسال هذا الثابت الرمزي إلى cout يكافئ نهاية السطر '\n' ولكن في هذه الحالة يفرغ مخزن الخرج المؤقت output buffer.

في المثال السابق قيمة n هي 66 . وهذه القيمة في الحقيقة خزنت في ذاكرة الحاسب في بتات bits متتالية كل منها يحتوي على الرقم 0 أو 1 . الحاسب يفسر أو يترجم هذه البتات المتتالية كعدد صحيح لأن المتغير أعلن عنه بأنه متغير للأعداد الصحيحة .

الإعلان declaration عن متغير هو جملة تعطي معلومات عن هذا المتغير لمترجم C++ . والإعلان يأخذ الشكل التالي :

```
type  variables
```

حيث أن type هي إسم لأحد أنواع المتغيرات في C++ فعلى سبيل المثال الإعلان التالي :

```
int n;
```

يخبر المترجم بأمرين : (1) إسم المتغير هو n و (2) هذا المتغير من نوع الأعداد الصحيحة int . كل

متغير يجب أن يكون له نوع . نوع المتغير يجب أن يخبر المترجم عن كيفية تخزين قيمة المتغيرات وإستخدامها . يمكن أن نصف النوع بمجموعة كل القيم التي يمكن أن تخصص للمتغير من هذا النوع . في بعض أجهزة الحاسب مجموعة القيم للنوع int تتكون من كل الأرقام في المدى من 32768- إلى 32767.

لغة ++C هي لغة برمجة موجهة . هذا يعني أن هذه اللغة جيدة في محاكاة المنظومات التي تتكون من أهداف متداخلة مثل منظومة التحكم في المطار . في مثل هذه المحاكاة ، الأهداف في المنظومة (الطائرات والناس والحقائب ... إلخ) تكون ممثلة بمتغيرات في برنامج الحاسب. لذلك فإن المتغيرات ينظر إليها على أنها الأهداف نفسها، ويمكن رؤيتها كوحدات قائمة بذاتها لها قدرات خاصة. في سياق هذا الشرح نقول أن الاعلان يخلق الهدف، والمتغير الذي يتم الإعلان عنه هو إسم الهدف .

نستطيع أن نرى تأثير الإعلان int n كالتالي :

```
n [ ?? ]
   int
```

الإعلان يخلق الهدف المبين هنا . إسم الهدف n ونوعه int . الصندوق يمثل ساحة الذاكرة المحددة لتخزين قيمة الهدف . علامات الإستفهام تدل على أن الهدف لم يأخذ قيمة إلى الآن.

التخصيص هي الطريقة الوحيدة التي بواسطتها يمكن تغيير قيمة الهدف . على سبيل المثال .

```
n = 66;
```

تغير قيمة n إلى 66 . ونستطيع أن نرى هذا التخصيص كما يلي :

```
n [ 66 ]
   int
```

في لغة ++C الإعلان يمكن أن يكون في أي مكان داخل البرنامج كما هو موضح في المثال التالي :

مثال 9.1

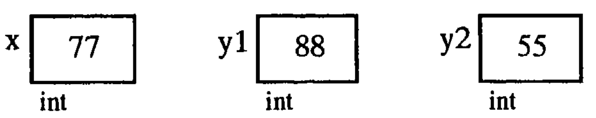
هذا المثال يوضح أن المتغير يمكن أن يعلن عنه في أي مكان في برنامج ++C

```
#include <iostream.h>
// This program illustrates variable declarations :
main ( )
{
    int x , y1;        // declares the variables x and y1
    x = 77 ;
```

```
    y1  =  88 ;
    int   y2   = 55 ;          // declares the variable y2 , initializing it to 55
    cout  <<  x  <<  ",  " <<  y1  <<   ",  "  <<  y2  <<  endl;
    return   0;
}
```

```
77 , 88 , 55
```

المتغير y2 تم الإعلان عنه وتحديد قيمة له بعد تخصيص قيمة لـ y1 . نستطيع أن نرى هذه الأهداف الثلاثة كما يلي :



لاحظ أن المتغير لا يمكن أن يستعمل قبل الإعلان عنه .

في هذا الكتاب نستخدم الحرف السميك في البرنامج لنؤكد على أجزاء البرنامج الموضحة في المثال . عندما تكتب أنت البرنامج بغرض التنفيذ فلا ضرورة لاستعمال الأحرف السميكة . المثال السابق يوضح أيضاً كيفية الإعلان عن أكثر من متغير واحد بداخل نفس عبارة الإعلان.

العبارة   int x, y1;

تعلن عن كل من x و y1 بأنهما متغيران من نوع الأعداد الصحيحة . بصفة عامة أي عدد من المتغيرات يمكن أن يعلن عنه بداخل نفس عبارة الإعلان لو أن كل المتغيرات المعلن عنها من نفس النوع . شكل كتابة هذا الأمر في الصورة العامة هو :

type var1, var2, ... , varN;

المتغيرات تكتب بعد نوعها ، والفصلات تفصل بين المتغيرات في القائمة .

7.1 الكلمات المفتاحية والمميزات

في أي لغة برمجة يتكون البرنامج من عناصر منفردة تسمى رموز (tokens) وتشمل هذه الرموز أسماء المتغيرات والثوابت والكلمات المفتاحية والمعاملات وعلامات الترقيم .

مثال 10.1

```
#include <iostream.h>
// A simple program to illustrate tokens :
main ( )
{
    int n = 66 ;
    cout << n << endl;
    return 0;
}
```

```
66
```

هــذا البرنــامج يبين 15 رمزاً هي : main ، (،) ، { ، int ، n ، = ، 66 ، ; ، cout ، >> ، endl ، return ، 0 و } . الرمــز n هو متـغير والرمـوز 66 ، 0 و endl ثوابت والرمـوز int و return كلمـات مفتاحية والرمـوز = و >> هـي مـعامـلات والرموز) و (و } و ; و { علامات ترقيم . يحتوي أول سطرين في البرنامج على توجيه أو ارشاد المعالج الأولي (preprocessor) . والتعليقات وهي ليست جزء حقيقي من البرنامج .

الكلمات المفتاحية تسمى أيضاً الكلمات المحجوزة أو المخصصة لأن هذه الكلمات محجوزة لأغراض معينة في لغة البرمجة ولا يمكن إستعمالها كأسماء لمتغيرات أو أي أغراض أخرى .

المميز (identifier) هو سلسلة من الحروف الأبجدية والعددية تبدأ بأحد الحروف الأبجدية.

يوجد 53 حرفاً أبجدياً : 52 حرفاً بالإضـافة إلى الشرطة السفلية ـ . يوجد 63 حرفاً أبجدياً وعددياً : 53 حرفاً أبجدياً و 10 أرقام (0, 1, 2, ...,9) . الكلمـات ( ) main ، int ، n ، cout و endl تعتبر مميزات ، وكذلك stack ، x1 ، y4 ، Lastname و the_day_after_tomorrow . لاحظ أن لغـة ++C حساسة لحالة الحرف : فهي تميز الحروف الكبيرة (Uppercase) من الحروف الصغيرة (lowercase) لذلك فـإن stack و Stack يعتبران مميزان مختلفان.

المميزات تستخدم لتسمية الأشياء مثل المتغيرات والدوال . في البرنامج السابق ، main إسم لدالة و int إسم لنوع بيانات و n و cout أسماء لمتغيرات و endl إسم ثابت . بعض المميزات مثل int تسمى كلمات مفتاحية لأنها جزء جوهري من لغة البرمجة نفسها . (الكلمات المفتاحية الـ 48 التي تعرف لغة ++C موضحة في الملحق B) . مميزات أخرى مثل n معرفة في البرنامج نفسه .

8.1 إعطاء قيمة إبتدائية للمتغير في أمر الإعلان

المتغير يمكن أن يأخذ قيمة إبتدائية "initialized" وذلك بإعطائه قيمة عند الإعلان عنه .

مثال 11.1 إعطاء قيم ابتدائية للمتغيرات

هذا البرنامج البسيط يوضح طريقتين لإعطاء قيم إبتدائية للمتغير بداخل أمر الإعلان عنه :

```
# include <iostream.h>
// This shows how to initialize variable as they are declared:
main ( )
{
     int  george = 44;
     int martha = 33 ;
     int sum = george + martha ;
     cout << george << " + " << martha << " = " << sum << endl;
     return 0 ;
}
```

```
44 + 33  =  77
```

المتغيران george و martha أخذا القيم الإبتدائية 44 و 33 بداخل أمر الإعلان عنهما . مع الإعلان عن المتغير sum فإن الجملة التعبيرية george + martha تحدد بالمجموع 44 + 33 وقيمة الناتج 77 خصصت للمتغير sum.

إعطاء قيم إبتدائية "initialization" هي تقريباً نفس طريقة التخصيص "assignment" . كل من الطريقتين يستعمل علامة التساوي " = " متبوعاً بجملة تعبيرية . الجملة التعبيرية تقدر أولاً ثم تخصص قيمة الناتج للهدف الموجود على يسار معامل التخصيص .

بصفة عامة ، من الأفضل إعطاء قيماً للمتغيرات عند الإعلان عنها .

إعطاء قيم إبتدائية يمكن أن يستخدم أيضاً عند الإعلان عن أكثر من متغير بعبارة إعلان واحدة كما يوضح المثال الآتي :

مثال 12.1 إعطاء قيم إبتدائية للمتغيرات :

```
# include <iostream.h>
// This shows how to initialize variables as they are declared:
main ( )
{
    int n1,  n2 = 55,   n3,   n4,   n5 = 44,   n6;
    cout << n2 << "," << n5 << endl;
    return 0;
}
```

```
55 , 44
```

كل الستة متغيرات من n1 إلى n6 أعلن عنهم بالنوع int ولكن متغيرين فقط n2 و n5 خصص لهما قيماً في جملة الإعلان .

بعض برامج الترجمة (compilers) (على سبيل المثال ++Borland C) سوف يعطي تحذير لو أن أي من المتغيرات لم يخصص له قيمة إبتدائية .

9.1 التخصصات المتسلسلة

التخصيص نفسه هو عبارة تعبيرية لها قيمة أو مقدار . قيمة الجملة التعبيرية :

```
x = 22
```

هي 22 . ومثل أي قيمة أخرى ، قيمة التخصيص يمكن أن تستخدم في تخصيص أخر :

```
y = (x = 22) ;
```

هذا هو التخصيص المتسلسل . بداية خصص القيمة 22 لـ x ثم بعد ذلك خصص القيمة 22 لـ y . التخصيص المركب عادة يكتب بدون أقواس

```
y = x = 22;
```

بصفة عامة ، قيمة التخصيص هي أخر قيمة خصصت .

مثال 13.1 التخصيصات المخفية

هذا المثال يبين كيفية إستعمال التخصيص بداخل الجمل التعبيرية :

```
# include <iostream.h>
// This shows that an assignment can be part of a larger expression:
main ( )
{
     int  m,  n;
     m = (n = 66)  +  9;        // (n = 66) is an assignment expression
     cout <<  m  << " ,  " <<  n  <<  endl;
     return 0;
}
```

```
75 , 66
```

التخصيص المركب يخصص أولاً القيمة 66 للمتغير n . وعند ذلك يحسب قيمة الجملة التعبيرية 9 + (n=66) التي هي 75 . عند ذلك يخصص هذه القيمة (75) للمتغير m.

التخصيصات المختصرة عادة يجب تجنبها . على سبيل المثال أول سطرين في البرنامج السابق يمكن أن يكتبا بطريقة أفضل كالآتي :

```
int n = 66 ;
int m = n + 9 ;
```

هذا يوضح أيضاً الطريقة المفضلة لإعطاء قيم إبتدائية للمتغيرات مع الإعلان عنها . توجد بعض الحالات للتخصيصات المختصرة تجعل البرنامج أسهل في القراءة . على سبيل المثال العبارة الوحيدة التالية أفضل من 8 عبارات منفصلة :

```
n1 = n2 = n3 = n4 = n5 = n6 = n7 = n8 = 65535;
```

سوف نرى أمثلة أخرى للتخصيصات المختصرة في الفصل الثالث .

التخصيص المتسلسل لا يمكن إستخدامه في إعطاء قيم إبتدائية عند الإعلان :

```
int x = y = 22 ;          // ERROR
```

سبب الخطأ في الجملة السابقة هو أن إعطاء قيم إبتدائية للمتغيرات ليست تخصيصات . هما متشابهان ولكن المترجم يتعامل مع كل منهما بطريقة مختلفة . الطريقة الصحيحة للجملة السابقة هي :

```
int x = 22,  y = 22 ;     //OK
```

10.1 الفاصلة المنقوطة Semiclan

في لغة ++C الفاصلة المنقوطة تستخدم كفاصل بين العبارات أو نهاية الجملة . كل جملة (أو عبارة) يجب أن تنتهي بالفاصلة المنقوطة . هذا يختلف عن اللغات الأخرى مثل Pascal التي تستخدم الفاصلة المنقوطة كفاصل للأوامر . لاحظ أن السطور التي تبدأ بالرمز # مثل :

```
# include <iostream.h>
```

لا تنتهي بالفاصلة المنقوطة لأنها ليست جملة من البرنامج ولكنها توجيهات للمعالجة الأولية .

ذكرنا في المقطع السابق أن جمل (أو أوامر) ++C يمكن النظر إليها كصيغة تعبير جبرية والعكس صحيح . الجمل التعبيرية يمكن أن تستخدم كعبارات (أو أوامر) منفردة . على سبيل المثال الجمل التالية مقبولة في لغة ++C .

```
x + y ;

22 ;
```

هذه الجمل لا تؤدي أي عمل ولا فائدة منها. ولكنها جمل مقبولة في لغة ++C . سوف ترى بعض الجمل المفيدة فيما بعد.

الفاصلة المنقوطة تعمل مثل المعامل في صيغة التعبير الجبرية . فهي تحول صيغة التعبير الجبرية إلى أمر. الفاصلة المنقوطة ليست معامل حقيقي لأنها لا تنتج قيمة . ولكن هذا التحويل يساعد في توضيح الفرق بين صيغة التعبير الجبرية (expression) والأمر (statment) .

11.1 شكل البرنامج

لغة ++C هي لغة بدون قيود : فليس لها متطلبات عن أماكن وضع عناصر البرنامج على السطر أو الصفحة . وبالتالي فإن المبرمج عنده حرية كاملة في تنسيق شكل البرنامج . ولكن المبرمجين من أهل الخبرة يعلمون أن مهمة الكتابة وتصحيح الأخطاء والحفاظ على نجاح البرنامج تكون أسهل كثيراً باستخدام طريقة متناسقة للبرمجة وسهلة القراءة . علاوة على ذلك فإن آخرين سوف يجدون أن برامجك تكون أسهل في القراءة لو أنك إتبعت المصطلحات القياسية لأسلوب البرنامج . إليك بعض القواعد البسيطة التي يتبعها معظم المبرمجين بلغة ++C :

- وضع كل التوجيهات include# في بداية ملف البرنامج .
- ضع كل جملة (أو أمر) على سطر جديد .
- إترك مسافة في بداية كل الجمل التي داخل نفس البلوك.

• إترك مسافة على جانبي كل معامل كالتالي n = 4 .

هذه القواعد تقريباً متبعة في كل مكان في هذا الكتاب .

قاعدة أخرى من المهم اتباعها وهي اختيار اسماء المتغيرات بدقة . إستعمل الأسماء القصيرة لتقليل فرصة الأخطاء المطبعية . ولكن أيضاً إختار الأسماء التي تصف ما تمثله المتغيرات . هذا يسمى شفرة توثيق ذاتي self-documenting code . تقريباً ، كل المبرمجين بلغة C++ يستخدمون الحروف الصغيرة في أسماء المتغيرات إلا إذا كان الإسم مكوناً من كلمات متعددة فيكون الحرف الأول من كل كلمة حرف كبير . على سبيل المثال :

```
char middleInitial;

unsigned maxUnsignedint;
```

هذه الأسماء أسهل في القراءة من middleinitial و maxunsignedint . كبديل لذلك فإن بعض المبرمجين يستعمل الشرطة الأفقية (underscore) كالتالي :

```
char  middle_initial;

unsigned max_unsigned_int;
```

12.1 أنواع الأعداد الصحيحة

الأعداد الصحيحة هي كل مجموعة الأرقام : 3- ,3 ,2- ,2 ,1- ,1 ,0 إلخ : الأعداد الصحيحة الموجبة (unsigned integer) هي الأعداد الصحيحة الغير سالبة :

0, 1 , 2 , 3 إلخ . لغة C++ بها تسعة أنواع من الأعداد الصحيحة :

عدد صحيح قصير بدون اشارة	unsigned short int
عدد صحيح بدون إشارة	unsigned int
عدد صحيح طويل بدون إشارة	unsigned long int
عدد صحيح قصير	short int
عدد صحيح	int
عدد صحيح طويل	long int
حرف	char
حرف بإشارة	signed char
حرف بدون إشارة	unsigned char

الفرق بين الأنواع التسعة في مدى القيم التي تسمح باستعمالها . مدى القيم يعتمد على نظام الحاسب المستعمل . على سبيل المثال في معظم أجهزة الحاسب الشخصية (PCs) . مدى الأعداد الصحيحة int يتراوح بين 32767 و 32768- بينما في معظم محطات التشغيل UNIX فالمدى يتراوح بين 2147483647 و 214783648- . المقطع " int " يمكن حذفه من أسماء الأنواع short int , long int , unsignel short int و unsignel int و unsignel long int .

البرنامج الذي في المثال التالي يطبع مدى كل نوع من أنواع الأعداد الحقيقية الموجودة على جهازك . الحدود القصوى SCHAR_MIN و LONG_MAX إلخ هي ثوابت مخزنة في الملف الرأسي <limit.h> لذلك أضيف توجيه المعالجة الأولى الآتي :

```
# include <limit.h>
```

الضروري لقراءة هذه الثوابت .

مثال 14.1 مدى أنواع الأعداد الصحيحة :

البرنامج التالي يطبع الحدود القصوى لمدى الأنواع المختلفة للأعداد الصحيحة :

```
# include <iostream.h>
# include <limits.h>
// prints the constants stored in limits.h :
main ( )
{
    cout  <<  "minimum char =  " << CHAR_MIN << endl;
    cout  <<  "maximum char =  " << CHAR_MAX << endl;
    cout  <<  "minimum short =  " << SHRT_MIN << endl;
    cout  <<  "maximum short =  " << SHRT_MAX << endl;
    cout  <<  "minimum int =  " << INT_MIN << endl;
    cout  <<  "maximum int =  " << INT_MAX << endl;
    cout  <<  "minimum long =  " << LONG_MIN << endl;
    cout  <<  "maximum long =  " << LONG_MAX << endl;
    cout  <<  "minimum signed char =  " << SCHAR_MIN << endl;
    cout  <<  "maximum signed char =  " << SCHAR_MAX << endl;
    cout  <<  "maximum unsigned char =  " << UCHAR_MAX << endl;
```

```
    cout  <<  "maximum unsigned short =  " << USHRT_MAX << endl;
    cout  <<  "maximum unsigned =  " << UINT_MAX << endl;
    cout  <<  "maximum unsigned long =  " << ULONG_MAX << endl;
   return 0;
}
```

```
minimum char = -128
maximum char = 127
minimum short = -32768
maximum short = 32767
minimum int = -2147483648
maximum int = 2147483647
minimum long = -2147483648
maximum long = 2147483647
minimum signed char = -128
maximum signed char = -127
maximum unsigned char = 255
maximum unsigned short = 65535
maximum unsigned = 4294967295
maximum unsigned long = 4294967295
```

هذا الخرج من محطة تشغيل UNIX . فهو يبين أن هذا النظام يحتوي على ست أنوا ع فقط للأعداد الحقيقية :

char	المدى من 128- إلى 127 (واحد بايت)
short	المدى من 32767- إلى 32767 (2 بايت)
int	المدى من 2147483648- إلى 2147483647 (4 بايت)
unsigned char	المدى من 0 إلى 255 (1 بايت)
unsigned short	المدى من 0 إلى 65535 (2 بايت)
unsigned	المدى من 0 إلى 4294967295 (4 بايت)

يمكنك أن تعلم أن الأعداد الصحيحة من نوع short على سبيل المثال تشغل 2 بايت (16 بت) على هذا الحاسب لأن المدى 32768- إلى 32767 يغطي $2^{16} = 65536$ قيمة ممكنة . (تذكر أن البايت = 8 بت وهي الوحدة القياسية لتخزين الحروف) .

عند تشغيل بورلاند ++C على جهاز حاسب شخصي فإن هذا البرنامج سينتج نفس المدى لكل الأنواع ما عدا int و unsigned ينتجان التالي :

المدى من 32768- إلى 32767 (2 بايت) int

المدى من 0 إلى 65535 (2 بايت) unsigned

13.1 المعاملات الحسابية البسيطة

المعامل (operator) هو رمز يؤثر على تعبير جبري واحد أو أكثر منتجاً قيمة يمكن تخصيصها لمتغير . سابقاً تقابلنا مع معامل الخرج >> ومعامل التخصيص = .

بعض المعاملات البسيطة هي المعاملات التي تؤدي العمليات الحسابية : + و – و / و % . وهذه المعاملات تعمل على أنواع الأعداد الحقيقية لتنتج عدد حقيقي آخر : m + n حاصل جمع m و n ، أو m - n حاصل طرح n من m ، أو m*n حاصل ضرب m في n ، أو m/n حاصل قسمة m على n ، أو m % n يعطي باقي خارج قسمة m/n . هذه الستة معاملات ملخصة في الجدول التالي وموضحة في المثال الذي يليه .

جدول 1.1 معاملات الأعداد الصحيحة الحسابية

المعامل	الوصف	مثال
+	جمع	m + n
–	طرح	m - n
–	سالب	- n
*	ضرب	m * n
/	قسمة	m / n
%	باقي القسمة	m % n

مثال 15.1 معاملات الأعداد الصحيحة

هذا المثال يوضح إستعمال المعاملات الحسابية الستة :

```
# include <iostream.h>
// Tests arithmetic operators :
main ( )
{
    int m = 38 ,  n = 5 ;
    cout  <<  m  <<  " + "  <<  n  <<  " = "  <<  (m + n)  <<  endl;
    cout  <<  m  <<  " - "  <<  n  <<  " = "  <<  (m - n)  <<  endl;
    cout  <<         " - "  <<  n  <<  " = "  <<  (- n)    <<  endl;
    cout  <<  m  <<  " * "  <<  n  <<  " = "  <<  (m * n)  <<  endl;
    cout  <<  m  <<  " / "  <<  n  <<  " = "  <<  (m / n)  <<  endl;
    cout  <<  m  <<  "%"    <<  n  <<  " = "  <<  (m%n)    <<  endl;
    return 0;
}
```

```
38 + 5 = 43
38 - 5 = 33
   - 5 = -5
38 * 5 = 190
38 % 5 = 3
```

لاحظ أن 7 = 38/5 و 3 = 38%5 . هاتين العمليتين تعطيان معلومات كاملة عن القسمة العادية لـ 38 على 5 حيث 7.6 = 5 ÷ 38 . الجزء الصحيح للنتيجة هو 7 = 5 ÷ 35 والجزء الكسري هو 0.6 = 5 ÷ 3 . خارج قسمة العدد الصحيح وهو 7 وباقي القسمة 3 يمكن تجميعهم مع المقسوم 38 والمقسوم عليه 5 في الصيغة الآتية:

$7 \times 5 + 3 = 38$

معاملات خارج قسمة العدد الصحيح وباقي القسمة تكون أكثر تعقيداً إذا كانت الأعداد الصحيحة ليست موجبة . بالطبع فإن المقسوم عليه يجب أن لا يكون صفراً . لكن لو أن أي من m أو n سالب عندئذ m/n و m%n يمكن أن يعطي نتائج مختلفة من حاسب إلى آخر . المطلب الوحيد هو

$q * n + r = m$

حيث q = m/n و r = m%n

على سبيل المثال 14- مقسومة على 5 تعطي 2.8- .

خارج القسمة الصحيح يمكن أن يقرب إلى 3- أو 2- لو أن الحاسب قرب خارج القسمة q إلى 3- عند ذلك فإن باقي قسمة العدد الصحيح يكون 1 . لكن لو أن جهاز الحاسب قرب q إلى 2- عند ذلك فإن r سوف تكون 4-.

مثال 16.1 قسمة الأعداد الصحيحة السالبة

هذا البرنامج يستعمل لتحديد كيفية معالجة الحاسب لقسمة الأعداد الصحيحة السالبة :

```
#include <iostream.h>
// Tests quotient and remainder operators :
main ()
{
    int m = -14 , n = 5 , q = m/n, r = m%n;
    cout << "m = " << m << endl;
    cout << "n = " << n << endl;
    cout << "q = " << q << endl;
    cout << "r = " << r << endl;
    cout << "q*n + r = " << " (" << q << ")*(" << n << ") + "
         << r << " = " << q*n + r << " = " << m << endl;
    return 0;
}
```

```
m = -14
n = 5
q = -2
r = -4
q*n + r = (-2) * (5) + -4 = -14 = -14
```

هذا يعطي نفس النتائج من كل من محطة التشغيل UNIX باستخدام المعالج الدقيق "processor" من نوع Motorola 68040 ومن الحاسب الشخصي باستخدام المعالج الدقيق من نوع Intel Pentium .

14.1 الملازمة واسبقية تنفيذ العمليات

لغة ++C غنية بالعديد من المعاملات . (الملحق c يحتوي على 55 منهم) . حيث أن التعبير الجبري يمكن أن يحتوي على أكثر من معامل لذلك من المهم أن نعلم ترتيب أو أسبقية تنفيذ هذه المعاملات . نحن معتادين على ترتيب المعاملات الحسابية العادية :

معاملات الضرب * و القسمة / والباقي % لها أسبقية عن معاملات الجمع + والطرح – أي أنها تنفذ أولاً . على سبيل المثال :

42 - 5*3

تنفذ كالآتي :

42 - (3*5) = 42 - 15 = 27

أكثر من ذلك ، كل المعاملات الحسابية لها أسبقية في التنفيذ عن معاملات التخصيص والخرج . على سبيل المثال الأمر

n = 42 - 3*5

سوف يخصص القيمة 27 للمتغير n . أولاً المعامل * يستدعى ليحسب 3*5 بعد ذلك المعامل – يستدعي ليحسب 15 - 42 بعد ذلك المعامل = يستدعي ليخصص القيمة 27 للمتغير n .

هذا هو جزء من الجدول c1 في الملحق C .

جدول 2.1 بعض معاملات لغة ++C

المعامل	الوصف	اسبقية التنفيذ	الملازمة	الحالة	مثال
-	سالب	15	يمين	إحادي	-n
*	الضرب	13	شمال	ثنائي	m*n
/	القسمة	13	شمال	ثنائي	m/n
%	باقي القسمة	13	شمال	ثنائي	m%n
+	الجمع	12	شمال	ثنائي	m+n
-	الطرح	12	شمال	ثنائي	m-n
<<	إزاحة للشمال	11	شمال	ثنائي	cout << n
=	التخصيص البسيط	2	يمين	ثنائي	m=n

هذا الجدول به 8 معاملات تستعمل مع متغيرات الأعداد الصحيحة . هذه المعاملات لها 5 مستويات مختلفة للأسبقية . على سبيل المثال معامل النفي الأحادي له مستوى أسبقية 15 ومعامل الضرب الثنائي * له مستوى أسبقية 13 عملية النفي تنفذ قبل عملية الضرب . لذلك فإن التعبير الجبري m*-n ينفذ كالتالي m*(-n) . معاملات التخصيص لها أقل أسبقية عن كل المعاملات الأخرى لذلك فهي عادة تتم (أو تنفذ) في الآخر.

العمود المسمى "الملازمة" يخبرنا ماذا يحدث عندما يكون عدة معاملات مختلفة لها نفس مستوى الأسبقية تظهر في نفس التعبير الجبري. على سبيل المثال + و - لهما مستوى الاسبقية 12 وملازمة إلى اليسار لذلك فإن المعاملات تنفذ من اليسار إلى اليمين . على سبيل المثال في التعبير الجبري:

8 - 5 + 4

أولاً 5 تطرح من 8 وبعد ذلك 4 تضاف إلى الناتج :

(8 - 5) + 4 = 3 + 4 = 7

العمود المسمى الحالة "Arity" مدون به إذا كان المعامل أحادي أو ثنائي . أحادي "unary" يعني أن المعامل ينفذ على طرف واحد . على سبيل المثال معامل الزيادة البعدية ++ هو أحادي ++n يعمل على متغير واحد n . الثنائي (binary) يعني أن المعامل ينفذ على طرفين . على سبيل المثال الجمع + هو ثنائي : حيث m + n ينفذ على متغيرين m و n .

15.1 معاملات الزيادة والنقصان :

لغة ++C بها خصائص كثيرة موروثة من لغة C من أهمها معامل الزيادة ++ ومعامل النقصان – – . هذه المعاملات تحول المتغير إلى صيغة جبرية مختصرة لأشكال معينة من أوامر التخصيص .

مثال 17.1 معاملات الزيادة والنقصان

هذا المثال يوضح كيفية عمل معاملات بالزيادة والنقصان

```
# include <iostream.h>
// Tests the increment and decrement operators:
main ( )
{
    int m = 44,     n = 66;
    cout << "m = " << m << " , n = " << n << endl;
    ++m;
    - - n ;
    cout << "m = " << m << " , n = " << n << endl;
```

```
        m++;
        n--;
        cout << "m = " << m << " , n = " << n << endl;
        return 0;
}
```

```
m = 44,   n = 66
m = 45,   n = 65
m = 46,   n = 64
```

كل من معامل الزيادة القبلي ومعامل الزيادة البعدي له نفس التأثير هنا : كل منهم يجمع واحد على قيمة m . أيضاً معامل النقصان القبلي n-- ومعامل النقصان البعدي --n له نفس التأثير : كل منهم يطرح واحد من قيمة n.

عند الإستخدام في جملة تعبيرية منفردة فإن m++ و ++m كل منهما يكافئ التخصيص التالي :

m = m + 1;

كل منهم يزيد قيمة m بمقدار 1 . أيضاً الجملة التعبيرية n-- و --n كل منهما يكافئ التخصيص التالي :

n = n - 1;

كل منهما ينقص قيمة n بمقدار واحد . (معامل الزيادة ++ يستخدم في الاسم "C++" لأنه يزيد "increment" لغة C الأصلية ولغة C++ يوجد بها كل شيء موجود في لغة C وزيادة).

على أي حال عند استعمال التعبير الجبري الفرعي (تعبير جبري داخل التعبير الجبري) فإن عملية الزيادة القبلية m++ تختلف عن عملية الزيادة البعدية m++ . الزيادة القبلية تزيد المتغير أولاً قبل استعماله في التعبير الجبري الأكبر بينما الزيادة البعدية تزيد قيمة المتغير بعد استعمال القيمة السابقة له بداخل التعبير الجبري الأكبر. حيث أن عملية الزيادة تكافئ عملية تخصيص مستقلة، فإنه في الحقيقة يوجد أمرين يتم تنفيذهما عند إستعمال عملية الزيادة في تعبير جبري فرعي : تخصيص الزيادة والأمر الشامل الأكبر . الفرق بين الزيادة القبلية والزيادة البعدية هو ببساطة الفرق بين تنفيذ التخصيص قبل أو بعد الجملة الشاملة .

مثال 18.1 معاملات الزيادة القبلية والزيادة البعدية

هذا المثال يوضح الفرق بين الزيادة القبلية والزيادة البعدية :

```
#include <iostream.h>
// Tests the increment and decrement operators:
main ( )
{
    int  m = 66 ,  n;
    n  = ++m;
    cout  <<  "m  =  "   <<  m  <<   " , n  =  "   <<  n  <<  endl;
    n = m++;
    cout  <<  "m  =  "   <<  m   <<   " , n  =  "   <<  n  <<  endl;
    cout  <<  "m  =  "  <<  m++  <<  endl;
    cout  <<  "m  =  "  <<   m   <<  endl;
    cout  <<  "m  =  "  << ++ m  <<  endl;
    return  0;
}
```

```
m  =  67,  n  =  67
m  =  68,  n  =  67
m  =  68
m  =  69
m  =  70
```

في التخصيص الأول زيادة قبلية لـ m لذلك قيمتها زادت إلى 67 ثم خصصت هذه القيمة للمتغير n. في التخصيص الثاني زيادة بعدية لـm لذلك القيمة 67 خصصت للمتغير n ثم بعد ذلك m زادت إلى 68 .

في أمر الخرج الثالث زيادة بعدية لـ m لذلك القيمة الحالية لـ m (68) وضعت في مجري الخرج وبعد ذلك m زادت إلى 69 . في آخر جملة خرج زيادة قبلية لـ m زادت إلى 70 أولاً وبعد ذلك وضعت هذه القيمة في مجرى الخرج.

إستعمال معاملات الزيادة والنقصان في التعبيرات الجبرية الفرعية ممكن أن يخدع ويجب إستعماله بحذر . على سبيل المثال فإن ترتيب تنفيذ التعبيرات الجبرية التي تحتوي على هذه المعاملات ليست معرفة في اللغة وبالتالي لا يمكن التنبؤ بنتيجتها .

مثال 19.1 عدم التنبؤ بترتيب تنفيذ التعبيرات

```
#include <iostream.h>
main ( )
{
    int n = 5,  x;
    x = ++n * - -n;
    cout << "n = " << n << ",x = " << x << endl;
    cout << ++n << " " << ++n << " " << ++n << endl;
}
```

```
n = 5 ,  x = 25
8 7 6
```

في التخصص للمتغير x ، أولاً n قد زادت إلى 6 وبعد ذلك نقصت إلى 5 قبل تنفيذ معامل الضرب وبذلك كان حساب قيمة x هو 5*5 . في آخر سطر ، نفذت التعبيرات الجبرية الفرعية من اليمين إلى اليسار . الملازمة اليسارية لمعامل الخرج » ليست لها أهمية لأنه لا يوجد معاملات أخرى لها نفس مستوى الأسبقية.

16.1 تعبيرات التخصيص المركبة

معاملات الزيادة والنقصان تختصر أنواع معينة من التخصيصات . لغة ++C تسمح أيضاً بجمع معامل التخصيص مع المعاملات الأخرى . الشكل العام لهذه التخصيصات المركبة هو .

variable op = expression

على سبيل المثال التخصيص المركب

```
n+ = 8;
```

له نفس التأثير مثل التخصيص البسيط

```
n = n + 8 ;
```

أنه ببساطة أضاف 8 إلى المتغير n.

مثال 20.1 معاملات التخصيص

هذا المثال يوضح كيفية إستعمال معاملات التخصيص المركبة:

```
#include <iostream.h>
// Tests combined operators:
main ( )
{
    int  n = 44;
    n += 9;
    cout <<  n  << endl;
    n -= 5;
    cout <<  n  << endl;
    n  *= 2;
    cout <<  n   << endl;
    return 0;
}
```

```
53
48
96
```

الأمر n + = 9 تضيف 9 إلى n والجملة n - = 5 تطرح 5 من n ، والجملة n * = 2 تضرب n في 2.

17.1 تجاوز الحد الأعلى والحد الأدنى للأعداد الصحيحة

الأعداد الصحيحة في الحاسب محدودة بخلاف الأعداد الصحيحة في الرياضيات البحتة. كما رأينا سابقاً كل نوع من الأعداد الصحيحة له قيمة عظمى وقيمة صغرى . لو أن قيمة المتغير زادت عن حدود نوع هذا المتغير فإننا نحصل على ما يسمى فائض حسابي (overflow) .

مثال 21.1 اختبار الفائض الحسابي

هذا البرنامج يوضح ماذا يحدث عندما تحدث إفاضة في المتغير من النوع القصير .

```
#include <iostream.h>
#include <limits.h>
// Tests for overflow for type short:
main ( )
{
    short n = SHRT_MAX - 1;
    cout << n++ << endl;
    cout << n++ << endl;
    cout << n++ << endl;
    cout << n++ << endl;
    return 0;
}
```

```
32766
32767
-32768
-32767
```

إن القيم تلف أو تدور حول النهايتين العظمى والصغرى 32767 و 32768- بمعنى آخر فإن قيمة الناتج عند إضافة 1 إلى 32767 هو 32768- . وهذا خطأ واضح.

معظم أجهزة الحاسب يتعامل مع الفائض الحسابي بهذه الطريقة . القيم تدور حول النهايات ، لذلك فإن الرقم الذي يأتي بعد النهاية العظمى هو النهاية الصغرى. وهذا أسوأ نوع للخطأ الذي يمكن أن يحدث في الحاسب لأنه عادة لا يوجد أي دليل خارجي لحدوث أي شيء خطأ. لحسن الحظ لغة C++ تحتوي على إمكانيات تساعد المبرمج للتغلب على هذه المشكلة كما سوف نرى فيما بعد .

الفائض الحسابي هو أحد أنواع الأخطاء التي تحدث عند تنفيذ البرنامج (run_time error) . مثال آخر أكثر شيوعاً هو القسمة على الصفر . وهذا النوع ليس مشكلة كبيرة لأنك تستطيع أن تعرفه عندما يحدث حيث أن البرنامج ينهار "crashes" . أما الفائض الحسابي فهو مثل النزيف الداخلي : فأنت ربما لا تدرك أنك في خطر مميت.

18.1 النوع الحرفي

في لغة ++C النوع الحرفي char هو أحد أنواع الأعداد الصحيحة . هذا يعني أن أي متغير من نوع char يمكن أن يستخدم في التعبير الجبري للأعداد الصحيحة تماماً مثل أي نوع للأعداد الصحيحة الأخرى، على سبيل المثال معاملات الأعداد الصحيحة تطبق على المتغيرات من النوع الحرفي char :

```
char   c  =  45;

char   d  =  2 * c - 7;

c+ = d % 3;
```

الإسم "char" هو مختصر لكلمة "character" . الإسم char يستخدم لأن المتغيرات من هذا النوع عندما تكون دخل أو خرج للبرنامج فإنها تفسر أو تفهم على أنها حروف . عندما يمثل الحرف دخل للبرنامج فإن النظام يقوم بتخزينه كشفرة أسكي ASCII code (انظر الملحق A) كقيمة للعدد الصحيح من النوع الحرفي char . عندما يكون المتغير الحرفي يمثل خرج للبرنامج فإن النظام (system) يرسل الحرف المقابل إلى مجرى الخرج . وهذا موضح في المثال القادم .

لغة ++C تحتوي على ثلاثة أنواع للأعداد الصحيحة كل منها له 8 بت (bits - 8) : char و signed char و unsigned char . ولكن نوعين فقط من هذه الأنواع الثلاثة مختلفين . النوع char يمكن أن يكون أي من signed char أو unsigned char تبعاً لجهاز الحاسب . إستخدام النوع char للحروف العادية . إستعمل النوع unsigned char في سلسلة الحروف القصيرة . النوع signed char غالباً غير واضح الإستعمال ولكنه ممكن أن يكون إختيار جيد لو أنك إحتجت أن تخزن كمية كبيرة من الأعداد الصحيحة القصيرة جداً التي لا تكون خرج للبرنامج بواسطة معامل الخرج القياسي >> .

مثال 22.1 الخرج الحرفي

هذا المثال يبين كيفية خروج المتغيرات الحرفية من البرنامج

```
#include  <iostream.h>
//  Tests output of type char :
main ( )
{
    char  c  =  64 ;
    cout  <<  C++  <<  " " ;          // prints  '@'  and increments c to 65
    cout  <<  C++  <<  " " ;          // prints  'A'  and increments c to 66
```

```
    cout  <<  C++  <<  " " ;            // prints  'B'   and increments c to 67
    cout  <<  C++  <<  endl ;           // prints  'C'   and increments c to 68
    cout  =  96;
    cout  <<  C++  <<  " " ;            // prints  ' , '  and increments c to 97
    cout  <<  C++  <<  " " ;            // prints  ' a '  and increments c to 98
    cout  <<  C++  <<  " " ;            // prints  ' b '  and increments c to 99
    cout  <<  C++  <<  endl ;           // prints  ' c '  and increments c to 100
    return 0;
}
```

```
@  A  B  C
,   a   b   c
```

أول جملة للخرج تطبع المتغير الحرفي C في مجرى الخرج . حيث أن هذا المتغير الحرفي له قيمة عددية 64 فإن خرج هذا المتغير يكون الحرف "@" . (شفرة الآسكي لهذا الرمز هي 64) . بعد ذلك فإن قيمة المتغير c تزداد مباشرة إلى 65 التي تسبب أن يكون الحرف "A" هو الخرج التالي . (شفرة الآسكي للحرف A هي 65). البرنامج يستمر بنفس الطريقة إلى نهايته . (لاحظ أن أجهزة الحاسب التي تستعمل الشفرة EBCDIC سوف يختلف خرجها عن الخرج المذكور هنا) .

شفرة الآسكي كاملة مبينة في الملحق A .

مثال 23.1 الحصول على شفرة الآسكي

```
#include  <iostream.h>
//  Tests output of type char:
main ( )
{
    char  c  =  ' A ' ;
    cout <<  C++  <<  "  "  <<  int (c)  <<  endl;     //  prints  ' A '  and  65
    cout <<  C++  <<  "  "  <<  int (c)  <<  endl;     //  prints  ' B '  and  66
    cout <<  C++  <<  "  "  <<  int (c)  <<  endl;     //  prints  ' C '  and  67
```

```
    return 0;
}
```

```
A 65
B 66
C 67
```

كما تم في تنفيذ هذا البرنامج فإن المتغير الحرفي c أخذ القيم 65 و 66 و 67 و 68 . حيث أن المتغيرات الحرفية تطبع كحروف لذلك فإن أول شيء طبع على كل سطر هو الحرف الذي له شفرة أسكي مخزنة في المتغير الحرفي c. لذلك تم طباعة الحروف A و B و C . نحن نستخدم int (c) لطباعة القيمة العددية للمتغير الحرفي c .

التعبير int (c) يسمى تحويل النوع "cast" . فهو يحول المتغير c من متغير حرفي إلى متغير عدد صحيح. وهذا يسمح لنا بطباعة شفرة الأسكي للحرف.

أسئلة للمراجعة

1.1 صف طريقتين لكتابة التعليقات comments في برنامج ++C.

2.1 ما هو الخطأ في التعليق التالي :

```
cout << "Hello, /* change? */ World.\n";
```

3.1 ما الذي يفعله الإعلان declaration ؟

4.1 ما الغرض من توجيهات أو إرشادات المعالجة الأولية ؟

```
#include <iostream>
```

5.1 هل هذا برنامج ++C مقبول أو صحيح ؟ وضح:

```
main ( ) {22;}
```

6.1 من أين أتي إسم لغة "++C" ؟

7.1 ما هو الخطأ في هذه الإعلانات :

```
int frist = 22 , last = 99 , new = 44 , old = 66;
```

8.1 ما هو الخطأ في هذه الإعلانات :

```
int x = y = 22;
```

9.1 ما هو الخطأ في هذا البرنامج :

```
main ( )
{ n = 22;
cout << n << endl;
}
```

10.1 أوجد قيمة كل من التعبيرات الجبرية الآتية إذا كان صحيحاً أو بين لماذا هو غير صحيح :

a. 37/(5%2)

b. 37/5/2

c. 37 (5/2)

d. 37%(5%2)

e. 37%5%2

f. 37 - 5 - 2

g. (37 - 5) 2

11.1 أوجد قيمة كل من التعبيرات الجبرية الآتية بفرض أن قيمة m = 24 و n = 7 في كل حالة :

a. m - 8 - n

b. m = n = 3

c. m % n

d. m % n++

e. m% ++n

f. ++m - n --

g. m + = n- = 2

12.1 حدد في كل مما يأتي المميزات الصحيحة ، إذا كان المميز غير صحيح وضح لماذا ؟

a. r2d2

b. H20

c. secondCousinonceRemoved

d. 2nBirthday

e. the_united_states_of_America

f. _TIME_

g. _12345

h. x(3)

i. cost_in_$

مسائل محلولة

13.1 ما هو خرج البرنامج التالي :

```
#include <iostream.h>
main ( )
{
      / /  cout << "Hello ,  World. \n" ;
}
```

هذا البرنامج ليس له خرج . الشرطتين المائلتين يحولوا جملة الخرج إلى ملاحظة .

14.1 ما الخطأ في البرنامج التالي :

```
#include <iostream.h>
/ /  This program prints "Hello , World." :
main ( )
{
     cout << "Hello , World. \n"
     return 0;
}
```

الفاصلة المنقوطة ناقصة من نهاية جملة الخرج .

10.1 أكتب أربع جمل مختلفة لـ C++ كل منهم يطرح واحد من متغير العدد الصحيح n .

```
n = n -1 ;
n - = 1 ;
-- n ;
n - - ;
```

16.1 أكتب جملة واحدة لـ C++ تطرح مجموع x و y من z وبعد ذلك تزيد قيمة y بمقدار واحد .

```
z - = (x + y ++);
```

17.1 أكتب جملة واحدة لـ C++ تنقص قيمة المتغير n . وبعد ذلك تضيفه إلى المتغير total .

```
total + = --n ;
```

18.1 في كل مما يأتي افترض أن m = 5 و n = 2 قبل تنفيذ الأمر. بين ما قيمة كل من m و n بعد تنفيذ بالأوامر التالية :

a. `m * = n ++ ;`

b. `m + = --n;`

a. n سوف تكون 3 و m سوف تكون 10

b. n سوف تكون 1 و m سوف تكون 6.

19.1 بين وصحح الخطأ في كل مما يلي :

a. `cout >> count;`

b. `m = ++n + = 2 ;`

a. مجرى خرج الهدف cout يحتاج إلى معامل الخرج <<

الأمر يجب أن يكون `cout << cout ;`

b. التعبير ++n لا يمكن أن يكون على يسار التخصيص

20.1 تتبع الأوامر التالية وبين قيمة كل متغير بعد تنفيذ كل أمر :

```
int x, y, z;
x = y = z = 6;
x * = y + = z- = 4;
```

أولاً الرقم 6 خصص للمتغيرات z و y و x . بعد ذلك المتغير z طرح منه 4 وأصبح 2 .

عندئذ المتغير y زاد بـ 2 وبذلك أصبحت قيمته 8 ، عندئذ المتغير x ضرب في 8 وأصبحت قيمته 48.

21.1 علـى معـظم محـطات التشـغيل UNIX يـكون مـدى نـوع الأعـداد الصحيحـة int مـن 2,147,483,648- إلـى 2,147,843,647- . كم عدد البايتات (bytes) التي يشغلها المتغير من هذا النوع في ذاكرة الحاسب .

المدى من 2,147,483,648 إلى 2,147,483,647 يغطي 4,294,967,296 قيمة . هذا الرقم هو 2^{32} . لذلك كل عدد صحيح يتطلب 32 بت (bits) التي هي عبارة عن 4 بايت من ذاكرة الحاسب.

22.1 ما هو الفرق بين الأمرين التاليين :

```
char  ch = ' A ';

char  ch = 65 ;
```

كل من الأمرين له نفس التأثير : يعلن عن ch أنه متغير حرفي ويخصص له القيمة 65 . حيث أن هذه القيمة هي شفرة الأسكي للحرف ' A ' ويمكن للثابت الحرفي أن يستخدم أيضاً في تخصيص الرقم 65 للمتغير ch .

23.1 ما هي الأوامر التي يجب أن تنفذ لإيجاد الحرف الذي له شفرة أسكي 100؟

```
char  ch = 100;

cout  << c ;
```

24.1 كيف تستطيع أن تحدد إذا كان النوع char هو نفس النوع signed char أو unsigned char على جهاز الحاسب ؟

نفذ برنامج مثل الذي في المثال 1.14 وقارن الثوابت char_max و schar_max و unchar_max .

مسائل محلولة في البرمجة

25.1 أكتب برنامج يطبع أول جملة من عنوان Gettysburg .

من الضروري أن كل الذي سوف نحتاج إليه هنا هو إستعمال مجموعة من جمل الخرج ترسل أجزاء جملة العنوان إلى الهدف cout .

```
#include <iostream.h>
// prints the frist sentence of the Gettysburg Address:
main ()
{
     cout  <<  "\tFourscore and seven years ago our fathers\n";
     cout  <<  "brought forth upon this continent a new nation,\n";
     cout  <<  "conceived in liberty, and dedicated to the\n";
     cout  <<  "proposition that all men are created equal .\n";
     return  0;
}
```

```
    Fourscore and seven years ago our fathers
brought forth upon this continent a new nation,
conceived in liberty, and dedicated to the
proposition that all men are created equal.
```

نستطيع أيضاً أن نفعل هذا بتسلسل أجزاء جملة العنوان مع جملة خرج واحدة إلى cout كالتالي:

```
cout  <<  "\tFourscore and seven years ago our fathers\n"
       <<  "brought forth upon this continent a new nation,\n"
       <<  "conceived in liberty, and dedicated to the\n"
       <<  "proposition that all men are created equal .\n";
```

لاحظ أن هذا أمر واحد مع فاصلة منقوطة واحدة .

إذا كنت ترغب في أن تكون سطور خـرج البـرنامج أطول (أو أقصـر) من الأجـزاء المفـردة المرسلة إلى مجرى الخرج فإنه ببساطة يمكن أن تضبط مكان حرف السطر '\n':

```
#include <iostream.h>
// prints the frist sentence of the the Gettysburg Address:
main ()
```

```
{
    cout << "\tFourscore and seven years ago our fathers" ;
    cout << "brought forth upon\nthis continent a new nation, " ;
    cout << "conceieved in liberty, and dedicated\nto the ";
    cout << "proposition that all men are created equal. \n" ;
    return o;
}
```

```
Fourscore and seven years ago our fathers brought forth upon this continent a
new nation, conceived in liberty, and dedicated to the proposition that all men are
created equal.
```

لا تنس أن تضع مسافة خالية بعد آخر كلمة في كل سطر لا ينتهي بحرف نهاية السطر.

26.1 أكتب برنامج يطبع مجموع وفرق وضرب وخارج قسمة وباقي قسمة متغيرين من الأعداد الصحيحة . خصص القيم 60 و 7 للمتغيرين.

بعد الإعلان عن المتغيرين m و n بأنهم من الأعداد الصحيحة وتخصيص القيم 60 و 7 لهم نستخدم جملة خرج واحدة لطباعة قيم المتغيرات وبعد ذلك جملة خرج واحدة لكل عملية من العمليات الخمسة :

```
# include <iostream.h>
// prints sum, difference, product, and quotient of given integers:
main ( )
{
    int m = 60, n = 7 ;
    cout << "The integers are "  << m << " and "  << n << endl;
    cout << "Their sum is         " << ( m + n) << endl;
    cout << "Their difference is  " << ( m - n) << endl;
    cout << "Their product is     " << ( m * n) << endl;
    cout << "Their quotient is    " << ( m / n) << endl;
    cout << "Their remainder is   " << ( m % n) << endl;
    return 0;
}
```

```
The integers are 60 and 7
Their sum is          67
Their difference is   53
Their product is      420
Their quotient is     8
Their remainder is    4
```

لاحظ أن خارج القسمة 8 وباقي القسمة 4 يناسب العلاقة المطلوبة لخارج قسمة وباقي قسمة العدد الصحيح : (4) + (7) (8) = 60 .

27.1 أكتب برنامج يطبع بلوك الحرف "B" مكونة من 6 * 7 من النجوم كالتالي :

```
*****
*   *
*   *
*****
*   *
*   *
*****
```

نستخدم جملة خرج واحدة لكل صف في بلوك الحرف

```
# includle <iostream.h>
// prints the block letter "B" in a 7 x 6 grid:
main ( )
{
    cout << "*****" << endl;
    cout << "*   *" << endl;
    cout << "*   *" << endl;
    cout << "*****" << endl;
    cout << "*   *" << endl;
    cout << "*   *" << endl;
    cout << "*****" << endl;
    return 0;
}
```

بدلاً من endl لكل خرج يمكن أن ننهي كل سلسلة حرف بين علامتي إقتباس بحرف نهاية '\n' كالتالي:

```
cout << "*****\n" ;
```

مسائل إضافية

28.1 تتبع الأوامر التالية وبين قيمة كل متغير بعد تنفيذ كل منهم

```
int x, y, z;

x = y = z = 5;

x * = y + = z - = 1 ;
```

29.1 في معظم الأنظمة مدى النوع unsigned char هو 0 إلى 255 كم عدد البايتات التي سوف يشغلها المتغير من هذا النوع في ذاكرة الحاسب؟

مسائل برمجة إضافية

30.1 أكتب ونفذ برنامج يطبع إسمك وعنوانك.

31.1 أكتب ونفذ برنامج على الحاسب يطبع الجملة الأولى من عنوان Gettysburg بحيث لا يزيد عن 40 حرف في السطر .

32.1 نفذ البرنامج الذي في المثال 11.1 على جهازك الشخصي. إستخدم الخرج لتحديد الأنواع المختلفة للأعداد الصحيحة وكم عدد البايتات المطلوبة لكل منهم.

33.1 غير البرنامج المبين في المثال 16.1 لترى كيفية تعامل جهازك الشخصي مع قسمة العدد الصحيح 20 على 7- . حاول أن تتنبأ بخارج القسمة وباقي القسمة. بعد ذلك نفذ برنامجك على الحاسب لترى إذا كنت على صواب:

34.1 أكتب ونفذ برنامج على الحاسب يطبع الحرف الأول من إسمك الأخير كبلوك لحرف في شبكة مكونة من 7 x 7 نجوم (stars).

35.1 أكتب ونفذ برنامج على الحاسب يطبع الأربع سطور الأولى من قصيدة شكسبير 18 :

Shall I compare thee to a summer's day?

Thou art more lovely and more temperate.

rough winds do shake the darling buds of May,

And summer's lease hath all too short a date.

36.1 لكى تعلم ماذا يفعل جهازك الشخصي في المتغيرات التي لم يخصص لها قيم في البداية أكتب ونفذ برنامج على الحاسب يحتوي على السطرين الآتيين :

```
int n;

cout << n << endl;
```

38.1 أكتب ونفذ برنامج على الحاسب يسبب فائض حسابي لمتغير من نوع int .

39.1 أكتب ونفذ برنامج على الحاسب مثل المثال 22.1 يطبع شفرة الأسكي للأرقام من 1 إلى 10 وآخر 5 حروف صغيرة . استخدم الملحق A لتختبر خرج البرنامج.

40.1 أكتب ونفذ برنامج على الحاسب مثل المثال 22.1 يطبع شفرة الأسكي لعشرة حروف متحركة كبيرة وصغيرة . استخدم الملحق A لتختبر خرج البرنامج.

إجابات أسئلة المراجعة

1.1 أحد الطرق هي استخدام تعليق لغة C القياسية :

```
/* like this */
```

الطريقة الأخرى هي استخدام تعليق لغة ++C :

```
// like this
```

الطريقة الأولى تبدأ بشرطة مائلة ونجمة وتنتهي بنجمة وشرطة مائلة . والطريقة الثانية تبدأ بشرطتين مائلتين وتنهي بنهاية السطر.

2.1 أي شيء بين علامتي الاقتباس سوف يطبع بما في ذلك التعليق المقصود .

3.1 الاعلان يخبر المترجم باسم ونوع المتغير الذي أعلن عنه . أيضاً يمكن تخصيص قيم للمتغير بداخل جملة الإعلان .

4.1 إنه يحتوي على الملف الرئيسي iostream.h في البرنامج . وهذا يحتوي على الإعلانات المطلوبة للدخل والخرج مثل معامل الخرج >>.

5.1 هذا برنامج ++C مقبول (أو صحيح) . فهو يحتوي على جملة واحدة : 22 وهذه هي جملة جبرية لأن أي ثابت مثل 22 هو تعبير جبري مقبول . البرنامج لا يقوم بعمل أي شيء .

6.1 الإسم يشير إلى لغة C ومعامل الزيادة ++ . الإسم يوحي بأن لغة سي++ متقدمة عن لغة سي.

7.1 الشيء الوحيد الخطأ في هذه الإعلانات هي كلمة new لأنها من الكلمات المفتاحية (keyword) . الكلمات المفتاحية هي كلمات محجوزة ولا يمكن استخدامها كأسماء لمتغيرات.

8.1 الطريقة الوحيدة لاستخدام علامة التساوي في الإعلان هي تخصيص قيمة لمتغير. التعبير الجبري x = y = 22 غير مقبول في جملة واحدة تحتوي على الإعلان والتخصيص معاً لأن المتغير y يكون على يمين أو علامة تساوي . التعبير الصحيح يكون الآتي :

```
int   x = 22,   y = 22;
```

9.1 يوجد خطأين . المتغير n لم يعلن عنه و cout استخدم بدون وجود عبارة التوجيه للملف <iostream.h> في البرنامج.

10.1 أ - 37/(5*2) تقدر بـ 3 = 37/10 .

ب - 37/5/2 تقدر بـ 3 = 7/2 = (37/5)/2 .

جـ - هذا غير صحيح لعدم وجود معامل بين 37 و (5/2).

العملية المرادة كانت 37*(5/2) التي تقدر بـ 18 = 37/2 .

د - 37%(5%2) تقدر بـ 0 = 37%1 .

هـ - 37%5%2 تقدر بـ 0 = 2%2 = (37%5) %2 .

و - 2 - 5 - 37 تقدر بـ 30 = 2 - 32 = 2 - (5 - 37) .

ز - هذا غير صحيح لعدم وجود معامل بين (5- 37) و 2

العملية المقصودة هي 2 * (5 - 37) التي تقدر بـ 64 = 2 * 32 .

11.1 أ - m - 8 - n تقدر بـ 9 = 7 - 16 = 7 - (8 - 24).

ب - m = n = 3 تقدر بـ 3 .

جـ - m % n تقدر بـ 3 = 7 % 24 .

د - m % n++ تقدر بـ 3 = 24%7 = 24%(7++) .

هـ - m % ++n تقدر بـ 0 = 24%8 = 24%(++7) .

و - ++m - n-- تقدر بـ 8 = 7 - 25 = (7--) - (24++) .

ز - m + = n- = 2 تقدر بـ 29 = 5 = 24+ - (2 = -7) = 24+ .

12.1 أ – r2d2 هي مميز مقبول (أو صحيح).

ب – H2O هي مميز مقبول .

جـ – SecondCourrsinonceRemoved هي مميز مقبول .

د – 2ndBirthday مميز غير صحيح لأن أول حرف رقم وهذا غير مقبول .

هـ – مميز صحيح .

و – مميز صحيح .

ز – مميز صحيح .

ح – x (3) مميز غير صحيح لأنه يحتوي على الأقواس (,) وهذا غير مقبول .

ط – cost _in_$ مميز غير صحيح لأنه يحتوي على علامة الدولار $.

الفصل الثاني

2

الأوامر الشرطية وأنواع الأعداد الصحيحة

Conditional Statements and Integer Types

كل البرامج التي ذكرت في الفصل الأول تنفذ على الحاسب بطريقة متتالية (متسلسلة) : كل أمر ينفذ مرة واحدة حسب ترتيبه في البرنامج. الأوامر الشرطية تعطي سهولة أكثر في إنشاء البرامج حيث أن بعض الأوامر يتوقف تنفيذها على تحقيق شرط (أو شروط) معين يتغير أثناء تشغيل البرنامج.

هذا الفصل يصف طريقة إستعمال أمر الشرط if وأمر الشرط else .... if وأمر الإختبار switch أيضاً يبين طريقة إدخال البيانات إلى البرنامج.

1.2 الدخل

في لغة ++C الدخل هو المناظر (أو المماثل) للخرج. البيانات الخارجة من تنفيذ البرنامج تمر في مجرى الخرج cout والبيانات الداخلة من لوحة المفاتيح إلى البرنامج أثناء التشغيل تمر في مجرى الدخل cin (تنطق "see-in") . الإسم مخصص "لوحدة إدخال" .

مثال 1.2 إدخال الأعداد الصحيحة

هذا برنامج بسيط يقرأ الأعداد الصحيحة الداخلة من لوحة المفاتيح :

```
main ( )
{
    int age;
    cout  <<  "How old are you: ";
    cin  >>  age;
    cout  <<  "In 10 years, you will be"  << age + 10 <<   " . \n";
}
```

```
How old are you: 19
In 10 years, you will be 29.
```

العدد المبين بالخط السميك في منطقة خرج البرنامج المظللة هو الدخل الذي تم بواسطة مستخدم البرنامج عن طريق لوحة المفاتيح.

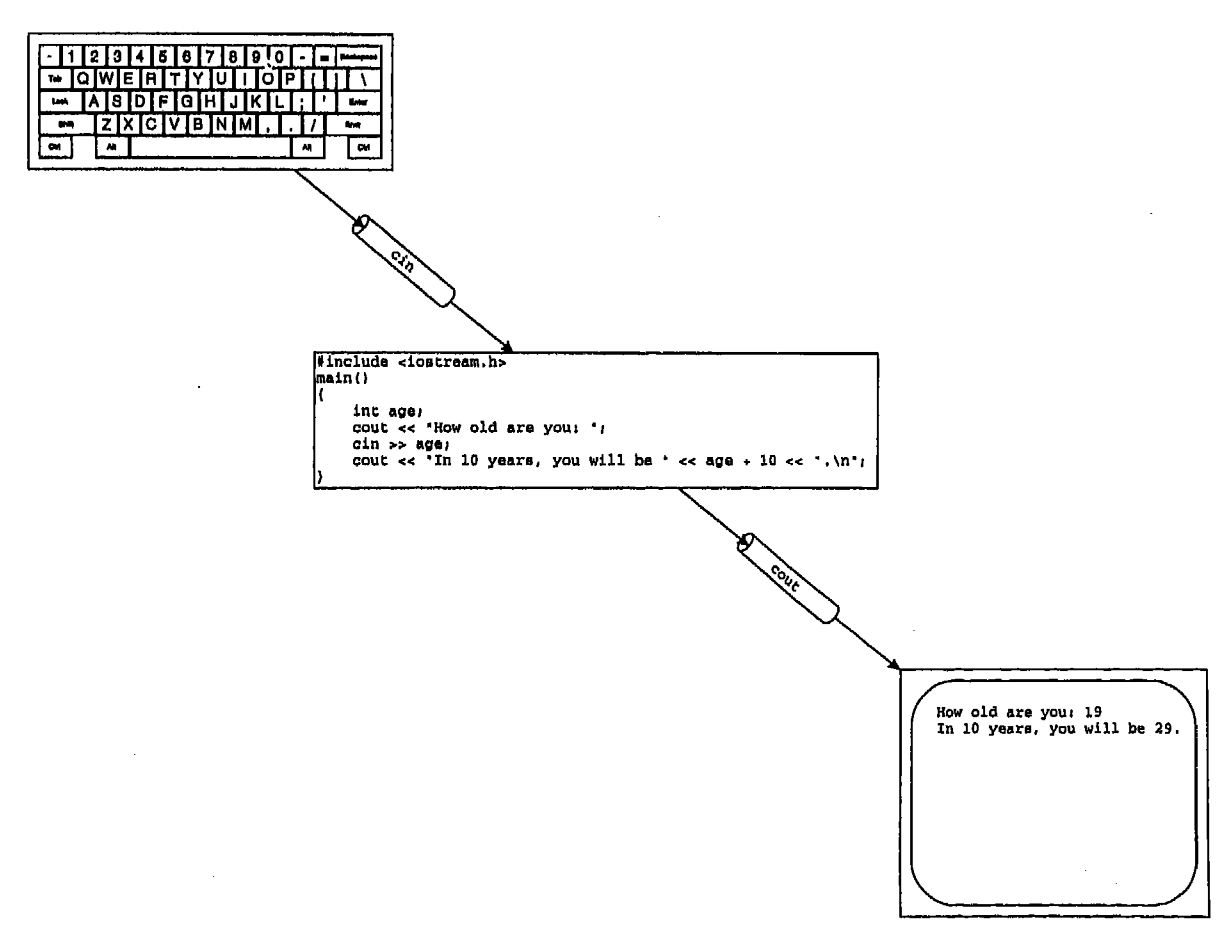
الرمز << هو معامل الاستخلاص ويسمى أيضاً معامل الإدخال. هذا المعامل يستخدم عادة مع مجرى الدخل cin والذي هو عبارة عن لوحة المفاتيح. لذلك عند تنفيذ الجملة التالية :

```
cin >>  age;
```

فإن البرنامج يتوقف منتظراً أن يدخل مستخدم البرنامج عدداً صحيحاً من لوحة المفاتيح. وعند إدخال هذا العدد فإنه يخصص للمتغير age ثم يستمر البرنامج في التنفيذ . لاحظ أن أمر توجيه المعالجة الأولية :

```
# include <iostream.h>
```

غير موجود في المثال 2.1 ولكنه مطلوب في أي برنامج يستخدم أي من cin أو cout . حيث أن كل برنامج في هذا الكتاب تقريباً يستخدم أي من cin أو cout فإننا سوف نفترض أنك تستطيع إضافة هذا التوجيه في بداية البرنامج. حذف هذا التوجيه من هذه الأمثلة ما هو إلا توفير حيز للطباعة . سوف تحذف أيضاً كلمة return في نهاية البرنامج الرئيسي ( ) main في كل الأمثلة التي سوف تأتي مستقبلاً. مجرى الدخل cin هو مناظر لمجرى الخرج cout . كل منهم في لغة ++C يمثل مجرى عبارة عن قناة تمر خلالها المعلومات. المعلومات التي تمر إلى داخل البرنامج أثناء تشغيله تمر خلال cin والمعلومات التي تخرج من البرنامج تمر خلال cout هذا يمكن تخيله كالتالي :



مثال 2.2 إدخال الحروف

```
main ()
{
      char frist, last;
      cout  <<  "Enter your initials:\n" ;
      cout  <<  "\tFirst name initial: ";
      cin  >> first;
      cout  <<  "\tLast name initial: ";
      cin  >> last;
      cout  << "Hello,   " << first << " .  " << last << " . !\n";
}
```

```
Enter your initials:
      First name initial: J
      Last name initial: H
Hello, J. H. !
```

هذا المثال يوضح طريقة قياسية لشكل الإدخال. أول سطر للخرج يذكر مستخدم البرنامج بالصورة العامة للدخل الذي يحتاج إليه، حيث يتبع ذلك مجموعة متتالية من طلبات الإدخال المحددة تسمى تذكيرات مستخدم البرنامج (user prompts) كل تذكير لمستخدم البرنامج يبدأ برمز ترك حقل خالي \t ويحذف رمز الانتقال إلى سطر جديد \n يجعل المؤشر cursor يظل على نفس السطر لإدخال رد المستخدم.

مثال 3.2 الإدخال المتعدد في نفس المجرى

يمكن قراءة أكثر من متغير بجملة إدخال واحدة :

```
main ()
{
      char first, last;
      cout  <<  "Enter your frist and last initials: " ;
      cin  >>  first  >> last;
```

```
    cout << "Hello, " << first << " . " << last << " .!\n";
}
```

```
Enter your first and last initials: JH
Hello,  J.  H. !
```

هذا المثال يبين أن مجرى الدخل cin يقرأ المتغيرات من اليسار إلى اليمين ، أي أن المتغير الذي في أقصى اليسار يقرأ أولاً .

حيث أن النوع الحرفي char هو نوع للأعداد الصحيحة لذلك فإن cin سوف يهمل الأماكن الخالية white spaces التي تلي المسافات الخالية blanks والحقول tabs ورمز الانتقال إلى سطر جديد عند قراءة الدخل ، لذلك فإن إدخال المتغيرات في هذا المثال يمكن أن يكون كالتالي :

```
Enter your first and last initials:   J    H
Hello, J.   H. !
```

لاحظ أن هذا يمنع معاملة المسافات الخالية مثل معاملة الحروف باستخدام معامل الادخال <<.

في الفصول القادمة سوف نرى طرق أكثر تخصص لإدخال الحروف.

2.2 عبارة if الشرطية

عبارة if الشرطية تسمح بتنفيذ بعض جمل البرنامج إذا تحقق شرط معين . الشكل العام لهذه العبارة الشرطية هو :

```
if (condition) statement ;
```

حيث أن condition هو تعبير جبري نتيجته عدد صحيح و statement هي أي جملة قابلة للتنفيذ . الجملة سوف تنفذ فقط لو أن الشرط condition كانت قيمته لا تساوي صفراً . (عند حساب قيمة أي تعبير جبري كشرط، فإنه إذا كانت قيمة هذا التعبير لا تساوي صفر فإنها تترجم أن الشرط حقيقي true وإذا كانت قيمة التعبير تساوي صفر فإنها تترجم إلى أن الشرط غير حقيقي false . لاحظ وجود الأقواس حول الجملة الشرطية.

مثال 4.2 إختبار قابلية القسمة

```
main ( )
{
      int  n,  d;
      cout  <<  "Enter two integers: ";
      cin  >>  n >>  d;
      if (n%d == 0)  cout  << n  <<  " is divisible by  "  <<  d  <<  endl;
}
```

```
Enter two integers : 24    6
24 is divisible by 6
```

هذا البرنامج يقرأ عددين من الأعداد الصحيحة ويختبر قيمة باقي خارج قسمة العددين n%d . في تنفيذ هذا البرنامج قيمة باقي خارج القسمة 24%6 هي صفر والتي تعني أن العدد 24 قابل للقسمة على العدد 6 .

المشكلة في البرنامج السابق أنه لا يقوم بعمل أي شيء إذا كان العدد n غير قابل للقسمة على العدد d :

```
Enter two integers :  24    5
```

لكي يقوم البرنامج بتنفيذ جملة أخرى عندما تكون الجملة الشرطية تساوي صفر فإننا نحتاج لاستعمال الجملة الشرطية if .... else .

3.2 الجملة الشرطية if .... else

الجملة الشرطية if ... else تقوم بتنفيذ أحد أمرين تبعاً لقيمة الشرط المحدد. وهذه الجملة الشرطية تأخذ الشكل الآتي :

if *(condition) statement1;*

else *statement2;*

حيث أن الشرط condition هو تعبير جبري نتيجته عدد صحيح والأمر الأول statement1 والأمر الثاني statement2 هما أي جمل أوامر مطلوب تنفيذها ، فالأول statement1 يتم تنفيذه إذا كانت القيمة العددية للشرط ليست صفراً والأمر الثاني statement2 يتم تنفيذه إذا كانت القيمة العددية للشرط تساوي صفراً .

مثال 5.2

هذا البرنامج هو نفس البرنامج الذي في المثال 4.2 مضافاً إليه كلمة else.

```
main ( )
{
int  n,  d;
cout  <<  "Enter two integers: ";
cin >> n >> d ;
if (n%d == 0) cout  << n <<  " is divisible by "  << d << endl;
else cout  <<  n  <<  " is not divisible by "  << d << endl;
}
```

```
Enter two integers :  24    5
24  is not divisible by 5
```

حيث أن باقي خارج قسمة 4 = 24%5 فإن الشرط (n%d == 0) غير صحيح أي أن قيمته تساوي صفر. وبالتالي يتم تنفيذ الأمر الثاني الذي يتبع جملة else .

شرط مثل (n%d == o) هو تعبير جبري قيمته العددية تفسر على أن هذا الشرط صحيح "true" أو غير صحيح "false" . في لغة ++C قيم الشرط تكون أعداد صحيحة :صفر 0 يعني أن الشرط غير صحيح "false" وأي قيمة أخرى صحيحة غير الصفر تعني أن الشرط صحيح "true" . نظراً لهذا التشابه فإن الشروط يمكن أن تكون تعبيرات جبرية من الأعداد الصحيحة. علي وجه الخصوص فإن نفس التعبير الجبري (n%d) يمكن أن يستخدم كشرط . عندما تكون قيمة الشرط لا تساوي صفراً (أي صحيح "true") فإن n تكون غير قابلة للقسمة على d ويجب أن نعكس جملتي الطباعة في المثال السابق ليكون لها معنى كالتالي :

مثـــال 6.2

```
main ( )
{
     int  n,  d;
     cout  <<  "Enter two integers: ";
```

```
        cin >> n >> d;
        if (n%d) cout << n << " is not divisible by " << d << endl;
        else cout << n << " is not divisible by " << d << endl;
}
```

```
Enter two integers : 24   5
24 is not divisible by 5
```

4.2 المعاملات النسبية

المثال التالي يحتوي على شرط في صورة أكثر بداهة

مثال 7.2 إيجاد العدد الاكبر في عددين صحيحين

هذا البرنامج يقوم بطباعة العدد الأكبر من بين عددين يتم إدخالهما إلى البرنامج .

```
main ()
{
        int m, n;
        cout << "Enter two integers: " ;
        cin  >> m >>n ;
        if (m > n) cout << m << endl;
        else cout << n << endl;
}
```

```
Enter two integers : 22   55
55
```

فـي هـذا البرنامج الشـرط هو (m > n) . إذا كـانت m أكبر من n فإن الشرط صـحيح "true" وتقدر قيمته بواحد وإلا فإن الشرط غير صـحيح وتقدر قيمته بصفر. لذلك فان قيمة m تطبع عندما تكون أكبر من n.

الرمز < هو واحد من المعاملات النسبية. ويسمى نسبي لأنه يقدر النسبة التي بين التعبيرين اللذين على جانبـيه ، على سـبيل المثال العلاقة 55 < 22 غير صـحيحة. والرمز < يسمى معامل "operater" لأنه عندما

يكون في تعـبير جـبري فإنه ينتج قيمة. على سبيل المثال عندما يجتمع الرمز < مع العدد 22 والعدد 55 في العلاقة 55 < 22 فإنه ينتج القيمة صفراً 0 وهذا يعني أن هذه العلاقة غير صحيحة "false".

يوجد ستة من المعاملات النسبية :

أقل من	<
أقل أو تساوي	<=
تساوي	==
أكبر من	>
أكبر من أو تساوي	>=
لا تساوي	!=

لاحظ علامتي التساوي == يجب أن يستخدما لإختبار المساواة . يوجد خطأ شائع بين المبرمجين الجدد بلغة C++ وهو استعمال علامة التساوي المفردة = . وهذا الخطأ يصعب اكتشافه لأنه صحيح من ناحية قواعد لغة C ++.

مثال 8.2 إيجاد العدد الاكبر من بين ثلاثة أعداد صحيحة

هذا البرنامج يقوم بطباعة أكبر عدد من بين ثلاثة أعداد يتم إدخالها إلى البرنامج :

```
main ( )
{
    int n1,  n2,  n3;
    cout  <<  "Enter three integers: ";
    cin  >>  n1  >>  n2  >>  n3;
    int max  =  n1;
    if  (n2  >  max)  max =  n2;
    if  (n3  >  max)  max =  n3;
    cout  <<  "The maximum is "  <<  max  <<  endl;
}
```

```
Enter three integers: 22  44  66
The maximum is 66
Enter three integers: 77  33  55
The maximum is 77
```

في أول تنفيذ للبرنامج تم إدخال العدد 22 للمتغير n1 والعدد 44 للمتغير n2 والعدد 66 للمتغير n3 . أولاً خصص العدد 22 للمتغير max . بعد ذلك خصص العدد 44 للمتغير max لأن العدد 44 أكبر من 22. في النهاية خصص العدد 66 للمتغير max لأن 66 أكبر من 44 وعند ذلك تم طباعة العدد 66.

في التنفيذ الثاني للبرنامج كانت قيمة المتغير n1 هي 77 و n2 هي 33 و n3 هي 55 . في البداية كان العدد 77 مخصص للمتغير max . وبعد ذلك لم تتغير قيمة المتغير max لأن 33 أصغر من 77 . في النهاية حيث أن 55 أصغر من 77 فإن قيمة المتغير max لم تتغير للمرة الثانية ولذلك تم طباعة القيمة 77 .

5.2 الأوامر المركبة

الأمر المركب هو مجموعة من الأوامر التي تنفذ كأنها أمر واحد . لغة ++C تعرف الجملة المركبة بوضع الأوامر المتتالية للجملة المركبة بين قوسين مجعدين. المثال التالي يحتوي على الأمر المركب التالي :

```
{
    int temp = x ;
    x = y ;
    y = temp;
}
```

الأقواس تحتوي على ثلاثة أوامر تكون بلوك . الجملة المركبة يمكن أن تستخدم في أي مكان آخر مثل أي جملة أخرى. (لاحظ أن برنامج الـ ++C كاملاً ـ كل شيء يأتي بعد كلمة ( ) main ـ يعتبر أمر مركب).

مثال 9.2 الترتيب

هذا البرنامج يقرأ عددين من الأعداد الصحيحة ويخرجهم تبعاً للترتيب التصاعدي :

```
main ()
{
    int  x,   y;
    cout  <<  "Enter two integer: ";
    cin   >>  x  >>  y;
    if  (x > y)  {
        int temp  = x ;
        x  =  y ;
```

```
        y = temp;
    }
    cout << x << " " << y << endl;
}
```

```
Enter two integers : 66 44
44 66
```

وضع الجملة المركبة مع الجملة الشرطية if يجعل كل الثلاثة أوامر الموجودة داخل البلوك (القوسين المعرجين) يتم تنفيذهم عند تحقق الشرط "true". هذه الأوامر الثلاثة على وجه الخصوص تقوم بعملية تبديل ، أي أنها تبدل قيم x و y . ويستخدم هذا غالباً في البرامج التي تقوم بترتيب البيانات. مثل هذا التبديل يحتاج إلى ثلاث خطوات بالإضافة إلى مخزن مؤقت سمي هنا temp . لاحظ أن المتغير temp تم الإعلان عنه داخل البلوك . وهذا يجعل المتغير محلي بالنسبة للبلوك أي أنه موجود أثناء تنفيذ البلوك. إذا كان الشرط غير صحيح و x ≤ y فإن المتغير temp لا وجود له. هذا مثال جيد لممارسة استعمال المتغيرات المحلية وهي التي تخلق فقط عند الحاجة إليها.

هذا المثال 9.2 ليس هو أفضل طريقة لحل المشكلة . إذا كنا نرغب في طباعة عددين بالترتيب التصاعدي فإنه يمكننا عمل ذلك مباشرة بدون استعمال المتغير temp:

```
if (x < y)   cout  << x  <<  " "  << y  <<  endl;
else cout  <<  y  <<  " "  <<  x  <<  endl;
```

الهدف من هذا المثال هو توضيح الجمل المركبة والإعلان عن المتغيرات المحلية .

6.2 كلمات اللغة المفتاحية

الكلمة المفتاحية key word في لغة البرمجة هي كلمة معرفة مسبقاً ومحجوزة لأداء غرض خاص في اللغة . يوجد 48 كلمة مفتاحية في لغة C++ وهي :

asm	continue	float	new	signed	try
auto	default	for	operator	sizeof	typedef
break	delete	friend	private	static	union
case	do	goto	protected	struct	unsigned
catch	double	if	public	switch	virtual
char	else	inline	register	template	void
class	enum	int	return	this	volatile
const	extern	long	short	throw	while

نحن رأينا من قبل الكلمات المفتاحية char و else و if و int و long و short و signed و unsigned . الـ 40 كلمة المفتاحية الباقية سوف توضح فيما بعد. الكلمات المفتاحية مثل if و else موجودة تقريباً في كل لغات البرمجة. الكلمات المفتاحية الأخرى مثل catch و friend هي كلمات مزيدة في لغة ++C. الكلمات المفتاحية الـ 48 في لغة ++C تحتوي على كل الكلمات المفتاحية الـ 32 في لغة C.

يوجد نوعين من الكلمات المفتاحية : الكلمات مثل if أو else والتي تستخدم في تركيب جمل البرنامج والكلمات مثل char و int وهي أسماء لأشياء في اللغة.

في بعض اللغات النوع الأول من الكلمات يسمي الكلمات المحجوزة reserved words والنوع الثاني يسمى المميزات القياسية standard identifiers .

7.2 الشروط المركبة

الشروط مثل n%d و x > y يمكن أن يجتمعا سوياً ليكونوا شروطاً مركبة. ثلاثة معاملات منطقية تستخدم لهذا الغرض، وهي معامل الجمع المنطقي && (and) ومعامل "أو" المنطقي || (or) ومعامل النفي المنطقي ! (not) . هذه المعاملات تعرف كالتالي :

قيمة الناتج تساوي 1 فقط إذا كان كل من p و q يساوي 1	p && q	&&
قيمة الناتج تساوي 1 إذا كان أي من p أو q تساوي 1	p || q	||
قيمة الناتج تساوي 1 إذا كانت p تساوي 0	!p	!

على سبيل المثال الشرط (n%d || x > y) يكون صحيح إذا كان أي من n%d لا تساوي صفر أو x أكبر من y (أو كليهما)، والشرط (x > y) ! يكافئ x <= y.

تعريف الثلاثة معاملات المنطقية يعطي عادة بجداول الحقيقة التالية :

p	q	p && q
1	1	1
1	0	0
0	1	0
0	0	0

p	q	p || q
1	1	1
1	0	1
0	1	1
0	0	0

p	! p
1	0
0	1

هذه الجداول تبين على سبيل المثال إذا كانت قيمة p تساوي 1 "true" وقيمة q تساوي 0 "false" فإن قيمة التعبير المنطقي p && q تكون صفراً وقيمة التعبير المنطقي p || q تكون 1 .

المثال التالي يحل نفس المشكلة التي تم حلها بالمثال 8.2 ما عدا أن هذا المثال يستخدم الشروط المركبة:

مثال 10.2 إيجاد القيمة العظمى من بين ثلاثة أرقام صحيحة

هذا المثال يستخدم الشروط المركبة لإيجاد القيمة العظمى من بين ثلاثة أرقام :

```
main ( )
{
    int a, b, c;
    cout << "Enter three integers: " ;
    cin >> a >> b >> c;
    if (a >= b && a >= c) cout << a << endl;
    if (b >= a && b >= c) cout << b << endl;
    if (c >= a && c >= b) cout << c << endl;
}
```

```
Enter three integers: 66 88 55
88
```

هذا المثال يختبر كل من الأعداد الثلاثة أيهما أكبر من أو يساوي العددين الآخرين. لاحظ أنه لا يوجد تعديلات في المثال 10.2 عن المثال 8.2 . الهدف فقط كان لتوضيح إستعمال الشروط المركبة.

وفيما يلي مثال آخر يستخدم شرط مركب :

مثال 11.2 الدخل السهل الاستخدام

هذا البرنامج يسمح للمستخدم بإدخال إما Y أو y للإجابة بنعم "yes".

```
main ( )
{
    char ans;
    cout << "Are you enrolled (y/n) : ";
    cin >> ans ;
    if (ans == 'Y' || ans == 'y') cout << "you are enrolled.\n";
```

```
	else cout << "you are not enrolled. \n";
}
```

```
Are you enrolled : N
you are not enrolled.
```

هذا البرنامج يحث المستخدم على الإجابة ويقترح إما أن تكون بحرف y أو n. ولكنه يقبل أي حرف ويستنتج منه أن إجابة المستخدم هي لا "no" إلا إذا كان الحرف المدخل هو Y أو y .

الشروط المركبة التي تستخدم المعاملات المنطقية && و || لن تنفذ الجزء الثاني من الشرط إلا عند الضرورة. وهذا يسمى القصر short-circuiting. كما تبين جداول الحقيقة، (p && q) سوف يكون غير صحيح "false" إذا كانت قيمة p غير صحيحة "false" . لذلك في هذه الحالة ليس هناك ضرورة لإيجاد قيمة q إذا كانت قيمة p غير صحيحة . بالمثل إذا كانت قيمة p صحيحة فإنه لا داعي لإيجاد قيمة q لتحديد نتيجة العلاقة (p || q) لأنها ستكون صحيحة.

القصر يمكن أن نراه من المثال التالي :

مثال 12.2 استعمال القصر في الشرط

هذا البرنامج يختبر قابلية القسمة للأعداد الصحيحة :

```
main ()
{
	int n, d;
	cout << "Enter two positive integers: ";
	cin >> n >> d;
	if (d > 0 && n%d == 0)   cout << d << " divides " << n << endl;
	else cout << d << " does not divide " << n << endl;
}
```

```
Enter two positive integers : 300 6
6 divides 300
```

```
Enter two positive integers : 300 7
7 does not divide 300
```

```
Enter two positive integers : 300  0
0  does not divide 300
```

في أول تنفيذ للبرنامج قيمة المتغير d كانت موجبة وباقي خارج القسمة n%d كانت صفراً لذلك فإن الشرط المركب كان حقيقي. في التنفيذ الثاني للبرنامج قيمة d كانت موجبة ولكن باقي خارج القسمة n%d ليس صفراً لذلك فإن الشرط المركب كان غير صحيح.

في التنفيذ الثالث للبرنامج قيمة d كانت صفراً لذلك فإن الشرط المركب تحدد مباشرة بأنه غير صحيح بدون إيجاد الجزء الثاني من الشرط المركب "n%d ==0" هذا القصر يمنع البرنامج من الإنهيار لأنه عندما تكون القيمة تساوي صفراً فإن التعبير n%d لا يمكن تقديره.

8.2 التعبيرات البولينية

التعبير البوليني هو شرط إما أن يكون صحيح أو غير صحيح. في المثال السابق التعبيرات d>0 و n%d ==0 و (d > 0 && n%d ==0) هي تعبيرات بولينية . كما رأينا فإن التعبيرات البولينية تقدر بقيم صحيحة int . الصفر يعني غير صحيح "false" وأي قيمة غير الصفر تعني حقيقي "true" .

حيث أن كل الأعداد الصحيحة التي لا تساوي صفر تعني أن الشرط صحيح "true" فغالباً تكون التعبيرات البولينية غير صريحة أو ممكن أن يساء تقديرها . على سبيل المثال الجملة

```
if (n) cout  <<  "n is not zero" ;
```

سوف تطبع على الشاشة العبارة n is not zero عندما تكون قيمة n ليست صفراً لأن ذلك يجعل التعبير البوليني (n) صحيح. مثال آخر أكثر واقعية :

```
if (n%d)  cout  <<  "n is not  a multiple of d" ;
```

هذه الجملة سوف تنفذ عندما تكون n%d ليست صفراً وهذا يحدث عندما تكون n لا تقبل القسمة على d بدون باقي لأن n%d هو الباقي من خارج القسمة.

إن حقيقة وجود قيم صحيحة للتعبيرات البولينية يمكن أن تؤدي إلى حدوث أشياء مذهلة في C++ كالتالي:

```
if  (x >= y  >= z) cout  <<  "max = x";    // ERROR !
```

واضح أن المبرمج يقصد أن يكتب ما يلي:

```
if (x >=  y  &&  y >= z)  cout  << "max = x";    //OK
```

المشكلة هي أن السطر الأول الذي هو خطأ أصلاً، هو صحيح من ناحية التركيب اللغوي لذلك فإن المترجم compiler لن يجد أي خطأ. في الحقيقة أن البرنامج يمكن أن ينفذ إذا لم يظهر المترجم أي أخطاء. هذا من أسوأ أنواع الأخطاء التي تحدث وقت التنفيذ لأنه لا يوجد دليل واضح يدل على وجود أي شئ خطأ.

مصدر الصعوبة هنا هو حقيقة أن التعبيرات البولينية لها قيم عددية. نفرض أن قيمة كل من x و y صفراً وأن قيمة z تساوي 1 . التعبير (x >= y >= z) يقدر من الشمال إلى اليمين . الجزء الأول x >= y صحيح "true" ويأخذ قيمة عددية 1 . عند ذلك يقارن بـ z وحيث أن قيمة z تساوي 1 والقيمة العددية للجزء الأول تساوي 1 فإن قيمة التعبير الكلي تكون صحيحة مع أنها في الحقيقة غير صحيحة !

يجب أن تتذكر هنا أن التعبيرات البولينية لها قيم عددية والشروط المركبة يمكن أن تسبب خدعة خطأ آخر يمكن أن يقع فيه المبرمجين المبتدئين في لغة ++C وهو استعمال علامة التساوي المفردة = عندما يكون المطلوب استعمال علامة التساوي المزدوجة == . على سبيل المثال.

```
if (x = 0)  cout << "x = o";  //  ERROR !
```

واضح أن المبرمج يقصد أن يكتب التالي :

```
if (x == 0)  cout << "x = 0"  ; // OK
```

الجملة الخطأ سوف تخصص 0 للمتغير x . وهذا يعني أن التعبير البوليني غير صحيح لذلك فإن الجملة cout لا يتم تنفيذها . لذلك حتى إذا كانت قيمة x الأصلية صفراً فإنها لا تطبع . الأسوأ من ذلك إذا كانت قيمة x الأصلية ليست صفراً فإنها سوف تغير إلى الصفر !

يعتبر مثل هذا الخطأ السابق من أسوأ الأخطاء وهو الذي يحدث وقت تنفيذ البرنامج ويكون من الصعب إكتشافه.

9.2 الشروط المتداخلة

الجمل الشرطية يمكن أن تستخدم في أي مكان مثل الجمل المركبة . لذلك يمكن أن تستخدم جملة شرطية بداخل جملة شرطية أخرى . وهذا يسمى تداخل (nesting) الجمل الشرطية . على سبيل المثال الشرط في المثال السابق يمكن أن يكرر كالتالي :

```
if   (d > 0)
     if  (n%d  == 0)
         cout  <<  d  <<  " divides "  <<  n  <<  endl;
     else
         cout  <<  d  <<  " does not divide "  <<  n  <<  endl;
else
     cout << d <<  " does not divide " << n << endl;
```

هنا فراغات كثيرة مستخدمة في إيضاح الجمل المنطقية المركبة ، وبالطبع فإن المترجم يهمل كل الفراغات والمسافات الخالية . لترتيب الجملة استخدمت قاعدة توافق else التالية :

وفق كل else مع آخر if ليس لها else

باستخدام هذه القاعدة ، يمكن للمترجم أن يحل الجمل الغامضة كالتالي :

```
if (a > 0)   if (b > 0)  ++ a; else if (c > 0)
if (a < 4)   ++ b; else if (b < 4) ++ c; else --a;
else if (c < 4) -- b ; else -- c ;  else a = 0 ;
```

وحتى تكون هذه الجمل أكثر سهولة في القراءة يمكن أن تكتب بطريقة أخرى كالتالي :

```
if (a > 0)
    if (b > 0)  ++ a;
    else
        if (c > 0)
            if (a < 4)   ++ b;
            else
                if (b < 4) ++ c;
                else --a;
        else
            if (c < 4)  -- b ;
            else  -- c ;
else
    a = 0 ;
```

أو كالتالي :

```
if  (a > 0)
    if (b > 0)  ++ a;
    else if (c > 0)
            if (a < 4)   ++ b;
            else if (b < 4)  ++ c;
            else --a;
    else if (c < 4)  -- b ;
    else  -- c ;
else
    a = 0 ;
```

مثال 13.2 حساب القيمة العظمى من بين ثلاثة أعداد

هذه طريقة أخرى لعمل ما تم عمله في مثال 8.2 ومثال 10.2 :

```
main ()
{
    int a, b, c, max;
    cout << "Enter three integers: ";
    cin >> a >> b >> c;
    if (a > b)
       if (a > c) max = a;          //  a > b and a > c
       else max = c;                // c >= a > b
    else
       if (b > c) max = b;          //b >= a and b > c
       else max = c;                //c >= b >= a
    cout << "The maximum is " << max << endl;
}
```

```
Enter three integers: 22 33 44
The maximum is 44
```

```
Enter three integers: 66 55 44
The maximum is 66
```

في أول تنفيذ للبرنامج إختبار الشرط (a > b) غير صحيح لذلك يتم تنفيذ الشرط (b > c) الذي يلي ثاني else وهو أيضاً غير صحيح لذلك يتم تنفيذ ثالث else والتي تخصص c للمتغير max . في ثاني تنفيذ للبرنامج ، الشرط (a > b) يكون صحيحاً وكذلك الشرط (a > c) أيضاً يكون صحيحاً لذلك خصصت a للمتغير max .

هذا البرنامج أفضل من البرنامج الذي في المثال 10.2 لأنه يختبر فقط شرطين مبسطين بدلاً من ثلاثة شروط مركبة . وبرغم ذلك يعتبر أقل منزلة لأن التعبيرات البولينية أكثر تعقيداً . التعليقات التي بداخل السطور

ضرورية لتوضيح التعبيرات المنطقية . الشروط المتداخلة معقدة بطبيعتها ، لذلك من الأفضل تجنبها إذا أمكن . استثناء من هذه القاعدة هي صورة خاصة للشروط المتداخلة حيث أن كل else تتبع مباشرة بـ if ما عدا آخر else . وهذا هو التركيب الشائع للجمل المنطقية لأنه يرتب تتابع البدائل بطريقة مبسطة . لتوضيح الجمل المنطقية فإن المبرمجين يضعوا عادة جمل if else في سطر واحد كما هو مبين في المثال التالي :

مثال 14.2

هذا البرنامج يحول درجات إختبار إلى ما يكافئها من الحروف الأبجدية

```
main ( )
{
     int score;
     cout  <<  "Enter the test score: ";
     cin  >>  score;
     if  (score  >  100)  cout  <<  "Error: score is out of range. ";
     else  if  (score  >=  90)  cout  <<  'A' ;
     else  if  (score  >=  80)  cout  <<  'B' ;
     else  if  (score  >=  70)  cout  <<  'C' ;
     else  if  (score  >=  60)  cout  <<  'D' ;
     else  if  (score  >=   0)  cout  <<  'F' ;
     else  cout  <<  "Error:  score is out of range.";
}
```

```
Enter the test score:  83
B

Enter the test score:  47
F

Enter the test score:  -9
Error: score is out of range.
```

المتغير score يتم إختباره خلال مجموعة من الشروط المتتالية إلى أن يتحقق أحد هذه الشروط أو الوصول إلى آخر else كما في التنفيذ الثالث للبرنامج.

10.2 الأمر switch

تركيب البدائل المتتابعة باستخدام else if يمكن أيضاً أن يتم باستعمال جملة الإختبار متعدد البدائل switch . وتركيبها كالتالي :

```
switch (expression) {
case constant1:    statementList1;
case constant2:    statementList2;



case constantN:    statementListN;
default : statementList;
}
```

الأمر switch يحدد قيمة التعبير expression ، فإذا كانت قيمة التعبير تساوي أي رقم ثابت لحالة case من الحالات فإن الأوامر التي في هذه الحالة سوف تنفذ. وإلا إذا كانت قيمة التعبير لا تساوي أي حالة فإن البرنامج ينفذ الأوامر التي في default . لاحظ أن قيمة التعبير يجب أن تكون عدد صحيح وكذلك الثوابت constants يجب أن تكون أعداد صحيحة (والتي تتضمن الحروف chars) .

مثال 15.2

هذا البرنامج يؤدي نفس أداء البرنامج الذي في المثال 14.2 :

```
main ()
{
    int score;
    cout << "Enter the test score: ";   cin >> score;
    switch (score/10)  {
        case 10:
        case  9: cout << 'A' << endl; break;
        case  8: cout << 'B' << endl; break;
        case  7: cout << 'C' << endl; break;
        case  6: cout << 'D' << endl; break;
        case  5:
```

```
        case  4:
        case  3:
        case  2:
        case  1:
        case  0: cout << 'F' << endl; break;
        default: cout << "Error: score is out of range. \n";
    }
}
```

في البداية البرنامج يقسم المتغير score على 10 . في ثاني تنفيذ للبرنامج حيث أن الدخل يكون 47 فإن قيمة التعبير (score /10) تقدر بـ 4 . هذه القيمة يتم تحديدها في قائمة الحالات case (أي أن case 4) ومن هذه اللحظة يتم تنفيذ كل الجمل الموجودة إلى break التالية. وهذا التدرج يتم على كل الحالات إلى أن نصل إلى الحالة رقم صفر و break التالية لها. هذه الظاهرة تسمى الإخفاق "fall through" .

11.2 معامل التعبير الشرطي

في لغة ++C يمكن كتابة جملة if ... else بطريقة مختصرة . وهذه الطريقة تسمى معامل التعبير الشرطي conditional expression وتستخدم الرموز ? و : في صورة ثلاثية خاصة :

condition ? expression1 : expression2

مثل أي معامل ، فإن هذه الصورة تضم التعبيرات وتنتج قيمة. هذه القيمة الناتجة إما أن تكون قيمة التعبير الأول expression1 أو التعبير الثاني expression2 تبعاً لحالة الشرط إذا كان صحيح أو غير صحيح. على سبيل المثال فإن الجملة التخصيصية :

```
min = x < y ? x : y;
```

سوف تخصص قيمة x إلى المتغير min إذا كانت x < y وإلا فإنها سوف تخصص قيمة y إلى المتغير min.

معامل التعبير الشرطي يستخدم بصفة عامة عندما يكون الشرط وكل من التعبيرين في صورة مبسطة جداً .

12.2 المجال SCOPE

مجال المميز هو الجزء من البرنامج الذي يمكن أن يستخدم فيه هذا المميز. على سبيل المثال المتغيرات لا يمكن أن تستخدم قبل الإعلان عنها ومجال هذه المتغيرات يبدأ عند الإعلان عنها. هذا موضح بالمثال التالي :

مثال 16.2 مجال المتغيرات

```
main ( )
{
     x = 11 ; // ERROR: this is not in the scope of x
     int x;
     {
          x = 22;    // OK: this is in the score of x
          y = 33;    // ERROR: this is not in the score of y
          int y;
          x = 44;   // OK: this is in the score of x
          y = 55;   //  ERROR: this is not in the score of y
     }
     x = 66;  // OK: this is in the score of x
     y = 77;  // ERROR: this is not in the score of y
}
```

مجال المتغير x يمتد من مكان نقطة الإعلان عنه إلى نهاية ( ) main . مجال المتغير y يمتد من نقطة الإعلان عنه إلى نهاية البلوك الداخلي الذي أعلن عنه فيه.

البرنامج يمكن أن يحتوي على أهداف متعددة بنفس الإسم إذا كان مجال كل متغير منفصل عن الآخر، وهذا موضح بالمثال التالي :

مثال 17.2 المجالات المتداخلة والمتوازية

```
int x = 11;                        // this x is global              هذا المتغير x متغير عام
main  ( )
{                            // begin scope of main ( )        بداية مجال البلوك main ()
   int x  =  22;
   {                     // begin scope of internal block       بداية مجال بلوك داخلي
      int x  =  33;
      cout  <<  "In block inside main ( ) : x  =  "  <<  x  <<  endl;
```

```
    }                    // end scope of internal block           نهاية مجال البلوك الداخلي
    cout << "In main ( ) : x = " << x << endl;
    cout << "In main ( ) : :: x = " << ::x << endl;
}                                   // end scope of main ( )       نهاية مجال الدالة main()
```

```
In block inside main ( ) : x = 33
In main ( ) : x = 22
In main ( ) : :: x = 11
```

في هذا البرنامج يوجد ثلاثة متغيرات مختلفة كل منها تسمى x. المتغير x الذي أخذ القيمة 11 هو متغير عام لذلك فإن مجاله يشمل كل البرنامج . المتغير x الذي قيمته 22 مجاله محدود في خلال الدالة main ( ) . حيث أن مجال المتغير الثاني داخل مجال المتغير الأول x فإن قيمة المتغير الأول تختفي داخل مجاله الدالة main ( ). المتغير الذي أخذ القيمة 33 مجاله محدود بالبلوك الداخلي داخل الدالة main ( ) ولذلك فهو يخفي كل من المتغير الأول والمتغير الثاني x داخل هذا البلوك. السطر الأخير في البرنامج يستخدم معامل تحليل المجال :: scope resolution operator للوصول إلى المتغير الذي لا يظهر في الدالة main ( ).

13.2 أنواع البيانات المتعددة enumeration

بالإضافة إلى أنواع البيانات (الأعداد) مثل int و char فإن لغة ++C تسمح لك بتعريف أنواع بيانات خاصة بك. هذا يمكن أن يتم بطرق متعددة أهمها هي التي تستعمل الطبقات classes كما هو موضح في الفصول من 8 إلى 14 وسوف نأخذ في الاعتبار هنا أبسط الطرق المستعملة.

نوع البيانات المتعددة هو نوع للأعداد الصحيحة يعرف بواسطة المبرمج بالشكل التالي:

enum *typename* { *enumeratorlist* };

حيث أن enum هي من الكلمات المفتاحية في لغة ++C و typename هي تعريف المميز الذي يحتوي على نوع الأسماء المعرفة، وقائمة البنود enumeratorlist هي قائمة المميزات التي تعرف ثوابت الأعداد الصحيحة. على سبيل المثال التعريف التالي هو تعريف لنوع البيانات المتعددة semester الذي يصف ثلاثة قيم يمكن أن يحصل عليها المتغير من هذا النوع.

```
enum semester {fall, spring, summer};
```

عندئذ يمكننا الإعلان عن المتغيرات من هذا النوع:

```
semester s1, s2;
```

ويمكننا استخدام هذه المتغيرات وأنواع القيم مثل الأنواع السابقة التعريف :

```
s1 = spring;

s2 = fall;

if (s1 == s2) cout << "Same semester. \n";
```

القيم الحقيقية المعرفة في قائمة البنود enumeratorlist تسمى البنود enumerators . وفي الحقيقة هي قيم صحيحة عادية . القيم fall و spring و summer المعرفة في النوع semester السابق يمكن أن تعرف كالآتي :

```
const int fall = 0;

const int winter = 1;

const int summer = 2;
```

القيم 0 ، 1 ، ... خصصت أتوماتيكياً عند تعريف النوع . هذه القيم التلقائية يمكن أن تلغي في قائمة البنود :

```
enum coin {penny = 1, nickel = 5, dime = 10, quarter = 25};
```

لو أن القيم الصحيحة خصصت لبعض البنود فقط عندئذ فإن البنود التالية تأخذ قيماً متتابعة. على سبيل المثال

```
enum Month {Jan = 1, feb, mar, apr, may, jun, jul, aug, sep, oct, nov, dec};
```

سوف يخصص الأرقام من 1 إلى 12 للإثنى عشر شهراً.

حيث أن البنود هي ببساطة ثوابت من الأعداد الصحيحة فإنه من الجائز أن تحتوي على عديد من البنود المختلفة بنفس القيمة :

```
enum Answer {no = 0, false = 0, yes = 1, true = 1, ok = 1};
```

هذا سوف يسمح بالشفرة التالية :

```
Answer ans;

:

:

if (ans == yes) cout << "you said it was o. k. \n";
```

والتي ستعمل كما هو متوقع. لو أن قيمة المتغير ans هي yes أو true أو ok (كل منهم =1) عند ذلك فإن الشرط سوف يكون حقيقي ويتم تنفيذ الخرج. لاحظ أن القيمة الصحيحة 1 دائماً تعني "true" في الشرط ويمكن أيضاً كتابة هذه الجملة الشرطية كالتالي :

```
if (ans)  cout  <<  "you said it was o.k. \n";
```

أنواع البيانات المنفردة عادة تعرف لجعل الشفرة أكثر تفسيراً أي أنها أسهل في الفهم. إليك مجموعة أمثلة نموذجية :

```
enum Boolean {false,  true};

enum Sex {female, male};

enum Day {sun, mon, tue, wed, thu, fri, sat};

enum Base {binary = 2,  octal = 8, decimal = 10, hexadecimal = 16};

enum Color {red, orange, yellow, green, blue, violet};

enum Rank {two, three, four, five, six, seven, eight, nine, ten, jack, queen, king,
        ace};

enum Suit {clubs, diamonds, hearts, spades};

enum Roman {I  =  1,  V =  5,  X =  10,  L = 50,  C= 100,  D = 500,  M  =  1000};
```

تعريفات مثل هذه يمكن أن تساعد في جعل الشفرة أكثر سهولة في القراءة لكن البيانات المتعددة يجب أن لا يفرط في استخدامها. كل بند في قائمة بنود يعرف مميز جديد. على سبيل المثال التعريف Roman السابق يعرف السبع مميزات I و V و X و L و C و D و M كثوابت من الأعداد الصحيحة لذلك فإن هذه الحروف لا يمكن إستخدامها لأي غرض آخر داخل مجال تعريفهم .

لاحظ أن البنود يجب أن تكون مميزات مقبولة . على سبيل المثال المميزات الآتية غير مقبولة :

```
enum Grade {F, D, C-, C, C+, B-, B, B+, A-, A};  // Error!
```

لأن الحروف '+' و '-' لا يمكن استخدامها في المميزات.

14.2 تحويلات الأعداد الصحيحة

في كثير من الحالات لغة ++C تسمح للمتغيرات من نوع معين أن تستخدم في مكان يكون المتوقع فيه هو نوع آخر. هذا يسمى تحويل النوع type conversion. أكثر الأمثلة شيوعاً لتحويل النوع هي من نوع الأعداد

الصحيحة إلى نوع آخر وهي التي تأخذ في الاعتبار هنا والتحويل من نوع الأعداد الصحيحة إلى الأعداد الحقيقية سوف يناقش في الفصل الثالث.

الفكرة العامة هي أن أحد أنواع الأعداد الصحيحة يمكن أن يستخدم في حين أن يكون المتوقع هو نوع آخر للأعداد الصحيحة إذا كان النوع المتوقع له رتبة أعلى. على سبيل المثال النوع char يمكن أن يستخدم في حين أن يكون المتوقع هو النوع int لأن int له رتبة (درجة) أعلى من char.

مثال 18.2 ترقية الأعداد الصحيحة

```
main ()
{
    char c = 'A';
    short m = 22;
    int n = c + m;
    cout << "n = " << n << endl;
}
```

```
n = 87
```

المتغير c من نوع char يأخذ العدد الصحيح 65 (شفرة الأسكي للحرف 'A') والمتغير m من نوع short يأخذ العدد الصحيح 22 . في جملة التخصيص n = c + m المتغيرات c و m لها أنواع صحيحة مختلفة لذلك فإن قيمهم 65 و 22 ترقي إلى النوع int قبل تخصيص القيمة الناتجة 87 للمتغير n.

ترقية الأعداد الصحيحة شائعة وعادة تحدث بدون ملاحظة . القاعدة العامة هي أن أي نوع للأعداد الصحيحة سوف يحول (يرقى) إلى النوع int حينما يكون هذا التحويل ضروري . استثناء لهذه القاعدة هي أنه في بعض برامج الترجمة "compilers" النوع int لا يغطي كل قيم النوع المرقى . في هذه الحالة نوع العدد الصحيح سوف يحول (يرقى) إلى النوع unsigned int . على سبيل المثال في بورلاند ++C مدى النوع unsigned int هو من 0 إلى 65,536 (إنظر المثال 14.1) الذي يزيد عن مدى نوع int (32768- إلى 32767) لذلك فإن هذا المترجم يحول (يرقي) unsigned short إلى unsigned int بدلاً من int.

حيث أن أنواع البيانات المتعددة هي أنواع للأعداد الصحيحة فإن تحويل (ترقية) الأعداد الصحيحة تطبق عليها أيضاً كما هو موضح في المثال التالي :

مثال 19.2 ترقية (تحويل) الأعداد الصحيحة

```
enum Color {red, orange, yellow, green, blue, violet};
main ( )
{
    Color x = blue;
    cout << "x = " << x << endl;
}
```

```
x = 4
```

في السطر الأخير قيمة x تحولت (ترقت) من نوع البيانات المحددة color إلى نوع الأعداد الصحيحة int قبل أن تدخل في مجرى الخرج.

أسئلة للمراجعة

1.2 أكتب جملة واحدة بلغة ++C تطبع العبارة "Too many" إذا زادت قيمة المتغير count عن 100.

2.2 ما الفرق بين الكلمة المحجوزة والمميز القياسي ؟

3.2 ما المقصود بالتعبير القصري "short-circuiting" وكيفية الاستفادة منه ؟

4.2 كيف تحدد قيمة التعبير التالي ؟

(x < y ? - 1 : (x == y ? 0:1);

5.2 ما المقصود بالتعبير "fall through" ؟

6.2 حدد إذا كان كل من الجمل التالية صحيح أو غير صحيح. إذا كان غير صحيح بين السبب.

a. !p || !q هي نفس ! (p||q)

b. !p هي نفس !!! p

c. p && (q||r) هي نفس p & & q ||r

7.2 ما الخطأ في الجمل التالية :

```
enum Semeter {fall, spring, summer};
enum Season {spring, summer, fall, winter};
```

8.2 ما الخطأ في الجملة التالية :

```
enum Frieends {"Tom", "Dick", "Harry",};
```

9.2 ما الخطأ في الجمل التالية :

```
if (x = 0) cout << x << " =0\n";
else cout << x << " ! = 0\n" ;
```

10.2 ما الخطأ في الجملة التالية :

```
if (x < y < z) cout << x << "<" << y << "<" << z << endl;
```

11.2 ما الخطأ في الجمل التالية :

```
a. cin << count;
b. if x < y  min = x
   else min = y;
```

12.2 ما الخطأ في الجمل التالية :

```
cout << "Enter n: ";
cin >> n;
if (n < 0)
     cout << "That is negative. Try again. \n";
     cin >> n;
else
     cout << "O.K. n = " << n << endl;
```

مسائل محلولة

13.2 كون التعبير المنطقي لتمثيل كل من الشروط التالية :

a. المتغير score أكبر من أو يساوي 80 وأقل من 90

b. المتغير answer إما أن يكون 'N' أو 'n'

c. n عدد زوجي ولكنه لا يساوي 8

d. المتغير ch هو حرف كبير

```
a.  (score >= 80 && score < 90);
b.  (answer == 'N' || answer == 'n');
c.  (n%2 == 0 && n != 8) ;
d.  (ch >= 'A' && ch <= 'Z') ;
```

14.2 ما الخطأ في الجمل الآتية :

```
if (x == 0)
    if (y == 0)  cout  << "x and y are both zero. \n";
else  cout << "x is not zero. \n" ;
```

واضح أن المبرمج يقصد طباعة الخرج الثاني "x is not zero. \n" إذا كان الشرط الأول (x == 0) غير صحيح بغض النظر عن حالة الشرط الثاني (y == 0) . لذلك فإن else وضعت متوافقة مع أول if . ولكن قاعدة توافق else تجعل else متوافقة مع الشرط الثاني الذي يعني أن الخرج "x is not zero. \n" سوف يطبع فقط عندما تكون x تساوي صفر و y لا تساوي صفراً.

يمكن إلغاء قاعدة " توافق else " بعمل أقواس :

```
if (x  == 0) {
    if  (y  == 0)  cout  << " x and y are both zero .\n";
}
else cout  <<  "x is not zero. \n" ;
```

الآن else متوافقة مع أول if وهذا ما يقصده المبرمج.

15.2 ما الفرق بين الجمل التالية :

```
if  (n  > 2) { if  (n  < 6)   cout  <<  "OK";} else cout  <<  "NG";
if  (n  > 2)  { if  (n  < 6)  cout  <<  "OK" ;  else cout  <<  "NG";}
```

في الجملة الأولى else متوافقة مع أول if . في الجملة الثانية else متوافقة مع ثاني if . إذا كانت n أقل من أو تساوي 2 فإن الجملة الأولى سوف تطبع NG بينما الجملة الثانية لا تفعل أي شئ. إذا كانت n أكبر من 2 وأقل من 6 فإن كل من الجملتين سوف يطبع OK . إذا كانت n أكبر من أو تساوي 6 فإن الجملة الأولى لا تفعل أي شئ بينما تطبع الجملة الثانية NG.

لاحظ أن هذه الجمل يصعب قراءتها لأنها لا تتبع المصطلحات القياسية. الجملة الأولى يجب أن تكتب هكذا:

```
if  (n > 2)  {
     if (n < 6) cout  <<  "OK";
}
else cout << "NG";
```

وجود الأقواس لازم هنا لإبطال قاعدة "توافق else " في الجملة السابقة else مقصود بها أن توافق أول if.

الجملة الثانية يجب أن تكتب هكذا :

```
if   (n > 2)
     if (n < 6)  cout << "OK";
     else  cout << "NG";
```

وجود الأقواس هنا غير لازم لأن else مقصود بها أن توافق ثاني if .

مسائل برمجة محلولة

16.2 أكتب ونفذ برنامج يقرأ عمر المستخدم ويقوم بطباعة الجملة "you are child" إذا كان عمره أقل من 18 ويطبع الجملة "you are an adult" إذا كان عمره أكبر من أو يساوي 18 وأقل من 65 ويطبع الجملة "you are a senior citizen." إذا كان عمره أكبر من أو يساوي 65.

سوف نستعمل هنا تركيب else if لأن جمل الخرج الثلاثة تعتمد على المتغير age الذي يكون في واحدة من الفقرات الثلاثة:

```
main ()
{
      int age;
      cout << "Enter your age: ";
      cin >> age;
      if (age < 18) cout << "you are a child. \n";
      else if (age < 65) cout << "you are an adult. \n" ;
      else cout << "you are a senior citizen. \n";
}
```

```
Enter your age: 44
you are an adult.
```

لو أن سريان البرنامج وصل إلى الشرط الثاني (age < 65) فإن هذا يعني أن الشرط الأول غير صحيح لذلك فإن $18 \le age < 65$. بالمثل لو أن سريان البرنامج وصل إلى ثاني else فهذا يعني أن كل من الشرطين السابقين غير صحيح لذلك فإن $age \ge 65$.

17.2 أكتب ونفذ برنامج يقرأ عددين صحيحين ويستعمل معامل التعبير الشرطي ليطبع إما "multiple" أو "not" تبعاً لحالة العددين إذا كان أحدهما يتكون من مضاعفات الآخر أو لا.

العدد الصحيح m يتكون من مضاعفات العدد الصحيح n إذا كان باقي خارج قسمة m على n صفراً . لذلك فإن الشرط المركب `m%n == 0 || n%m == 0` يستعمل لإختبار أي من العددين يتكون من مضاعفات الآخر.

```
main ()
{
      int m, n;
      cin >> m >> n;
      cout << (m%n == 0 || n%m == 0 ? "multiple" : "not") << endl;
}
```

```
30 4
not

30 5
multiple
```

قيمة التعبير الشرطي سوف تكون إما "multiple" أو "not" تبعاً لحالة الشرط المركب إذا كان صحيح أو غير صحيح. لذلك فإن إرسال التعبير الشرطي كاملاً إلى مجرى الخرج سوف ينتج النتيجة المطلوبة.

18.2 أكتب ونفذ برنامج يقوم بعمل آلة حاسبة بسيطة . البرنامج يقرأ عددين صحيحين وحرف. البرنامج يطبع مجموع العددين إذا كان الحرف هو + أو الفرق بين العددين إذا كان الحرف هو - أو حاصل ضرب العددين إذا كان الحرف هو * أو خارج قسمة العددين إذا كان الحرف هو / أو باقي خارج القسم إذا كان الحرف هو % . استعمل عبارة switch.

الحرف الذي يمثل نوع العملية (جمع أو طرح أو ...) يجب أن يكون هو الذي يتحكم في عبارة switch:

```
main ()
{
    int x, y;
    char op;
    cout << "Enter two integers: ";
    cin >> x >> y;
    cout << "Enter an operator: ";
    cin >> op;
    switch (op) {
        case '+' : cout << x + y << endl;   break;
        case '-' : cout << x - y << endl;   break;
        case '*' : cout << x * y << endl;   break;
        case '/' : cout << x / y << endl;   break;
        case '%' : cout << x % y << endl;   break;
    }
}
```

```
Enter two integers:  30   13
Enter an operator: %
4
```

فـي كـل من الحالات الخمـسـة نطبع قيمة العملية الرياضية المناظرة لكل حالة وبعد ذلك نخرج خارج بلوك swich .

19.2 أكتب ونفذ برنامج يلعب لعبة "Rock, paper, scissors" . في هذه اللعبة، كل من اللاعبين يقول في وقت واحد إما "rock" أو "paper" أو "scissors" . الفائز هو الذي يكون إختياره يتغلب على إختيار الآخر. القواعد هي : paper هـي التي تـغلب (تـخـفي) rock و rock تغلب (يـكسـر) scissor و scissor تغلب (يقطع) paper . استعمل أنواع البيانات المتعددة للاختيارات والنتائج .

في البداية سوف نعرف نوعين للبيانات المتعددة هي choice و Result. ثم نعلن عن المتغيرات choice1 و choice2 و result ونستخدم المتغير n لإدخال الاختيارات :

```
enum Choice  { rock,  paper,  scissors } ;
enum Result  { player1,  player2,  tie } ;
main  ( )
{
     int  n;
     choice choicel,  choice2;
     Result result;
     cout  <<  "Choose rock  (0) , paper  (1) , or scissors (2)  : \n";
     cout  <<  "Player #1: ";
     cin  >>  n;
     choice1 = choice (n);
     cout  <<  "Player # 2: " ;
     cin  >>  n;
     choice2 = choice (n);
     if  (choice1  ==  choice2) result = tie;
     else if (choice1  ==  rock)
          if (choice2 ==  paper) result  = player2;
          else result  = player1;
```

```
        else if (choice1  ==  paper)
             if (choice2  ==  rock) result  =  player1;
             else result = player2;
        else // (choice1  ==  scissors)
             if // (choice2  ==  rock)   result  =  player2;
             else result = player1;
        if (result  ==  tie)  cout  <<  "\tYou tied. \n";
        else if (result  ==  player1)  cout  <<  "\tplayer #1 wins. \n";
        else cout  <<  "\tplayer  #2 wins. \n";
}
```

```
choose rock (0) , paper (1) ,  or scissors (2) :
Player  #1: 1
Player  #2: 1
you tide.
```

```
choose rock (0) , paper (1) ,  or scissors (2) :
Player  #1:  2
Player  #2:  1
player  #1 wins.
```

```
choose rock (0) , paper (1) ,  or scissors (2) :
Player  #1:  2
Player  #2:  0
player  #2 wins.
```

باستعمال مجموعة متتالية من جمل if المتداخلة يمكننا أن نغطي كل الإحتمالات.

20.2 أكتب ونفذ برنامج يقوم بحل معادلات الدرجة الثانية .

معادلة الدرجة الثانية هي معادلة في الصورة $ax^2 + bx + c = 0$ حيث أن كل من a و b و c معاملات معلومة و x مجهول. المعاملات هي أعداد حقيقية لذلك يجب أن يعلن عنها بالنوع float أو double .

حيث أن معادلات الدرجة الثانية لها حلين لذلك سوف نستخدم المتغيرين x1 و x2 لهذه الحلول . وسوف يعلن عن عذه المتغيرات بالنوع double لتجنب عدم الدقة من الخطأ التراكمي.

حل معادلة الدرجة الثانية يعطي بالصورة الآتية :

$$x = \frac{-b \pm \sqrt{b^2 - 4ac}}{2a}$$

ولكن هذه الصورة لا تطبق إذا كانت a صفراً لذلك يجب التأكد من هذا الشرط أولاً . هذه الصورة أيضاً لا تصلح (في الاعداد الحقيقية) إذا كان المقدار الذي تحت الجذر سالب . هذا المقدار b^2 - 4ac يسمى مميز discriminant التربيع وسوف نعرف هذا المميز بالمتغير d ونتأكد من إشارته.

```
#include <iostream.h>
#include <math.h>       // needed for the sqrt () function
// This solves the equation  a*x*x + b*x + c == 0:
main ( )
{
    float  a,  b,  c;
    cout  <<  "Enter coefficients of quadratic equation: ";
    cin  >>  a  >>  b  >>  c;
    if  (a  == 0)   {
        cout  <<  "This is not quadratic a equation: a == 0\n";
        return 0;
    }
    cout  <<  "The equation is :  <<  a  <<  "x ^2 + " << b
        <<  "x + "  <<  c  <<  "  =  0\n";
    double  d,  x1,  x2;
    d  =  b*b  -  4*a*c;  // the discriminant
    if  (d < 0)   {
        cout  <<  "This equation has no real solutions: d  <  0\n;
        return  0;
    }
```

```
    x1 = ( -b + sqrt (d) ) / (2*a);
    x2 = ( -b - sqrt (d) ) / (2*a);
    cout << "The solutions are: " << x1 << ", " << x2 << endl;
}
```

```
Enter coefficients of quadratic equation: 2 1 -6
The equation is: 2x^2 + 1x + -6 = 0
The solutions are : 1.5, -2

Enter coefficients of quadratic equation: 1 4 5
The equation is: 1x^2 + 4x + 5 = 0
This equation has no real solutions : d < 0

Enter coefficients of quadratic equation: 0 4 5
This is not a quadratic equation: a == 0
```

لاحظ كيفية إستعمال العبارة return داخل الجمل الشرطية لإنهاء البرنامج إذا كانت قيمة أي من a تساوي صفر أو d سالبة. البديل هو إستعمال جمل else بعد كل if .

مسائل إضافية

21.2 أكتب جملة تخصيص واحدة تستخدم معامل التعبير الشرطي لتخصيص القيمة المطلقة للمتغير x إلى المتغير absx .

22.2 كون تعبير منطقي يمثل كل من الشروط الآتية :

a. المتغير weight أكبر من أو يساوي 115 وأقل من 125

b. المتغير ch إما أن يكون 'Q' أو 'q'

c. المتغير x هو عدد زوجي ولكنه لا يساوي 26

d. المتغير donation تكون قيمته من 1000 إلى 2000 أو قيمة المتغير guest تساوي واحد

e. المتغير ch هو حرف صغير أو حرف كبير

23.2 كون جدول الحقيقة لكل من التعبيرات المنطقية الآتية مبيناً قيمة كل منها (0 أو 1) في كل الإحتمالات الأربعة للمتغيرات p و q .

a. !p || q

b. p && q || !p && !q

c. (p ||q) && ! (p && q)

24.2 إستعمل جداول الحقيقة لتحديد إذا كان كل من التعبيرين المنطقيين في كل حالة من الحالات الآتية متكافئين أم لا.

a. !p && !q و ! (p && q)

b. p و !! p

c. p || q و !p || q

d. (p && q) &&r و p && (q && r)

e. (p || q) && r و p || (q && r)

25.2 أكتب جملة واحدة بلغة ++C تطبع العبارة "too many" إذا زادت قيمة المتغير count عن 100 باستعمال :

a. عبارة if

b. معامل التعبير الشرطي

مسائل برمجة إضافية

26.2 أعد كتابة برنامج "Hello world" بحيث يقرأ الثلاثة حروف الأولي من اسم مستخدم البرنامج ويطبعها بدلاً من كلمة "world!" على سبيل المثال لو أن مستخدم البرنامج أدخل الحروف R و W و D فإن خرج البرنامج يكون

Hello, R. W. D.

27.2 أكتب ونفذ برنامج يقرأ أربعة أعداد صحيحة ويطبعهم في عكس ترتيب إدخالهم.

28.2 أكتب ونفذ برنامج يقرأ أربعة أعداد صحيحة ويطبع العدد الأصغر والعدد الأكبر . إستعمل الجمل الشرطية كما في المثال 8.2 .

29.22 أكتب ونفذ برنامج يقرأ التقدير A أو B أو C أو D أو F ويطبع "excellent" أو "good" أو "fair" أو "poor" أو "failure" . إستعمل الجملة الشرطية else if .

30.2 أكتب ونفذ برنامج يطبع جداول الحقيقة لكل من التعبيرات المنطقية التي في المسألة 23.2 .

31.2 أكتب ونفذ برنامج يطبع جداول الحقيقة التي تحقق إجاباتك على المسألة 24.2 .

32.2 في سنة 1993 في الولايات المتحدة كان بيان قيمة الضرائب لدافعي الضرائب من ذوي الحالات المقررة كالتالي :

جدول 2.1

If the amount on Form 1040, Line 37, is: Over—	But not over—	Enter on Form 1040, lime 38	of the amount over—
$0	$22,100	15%	$0
22,100	53,500	$3,315.00 + 28%	22,100
53,500	115,000	12,107.00 + 31%	53,500
115,000	250,000	31,172.00 + 36%	115,000
250,000	-------	79,772.00 + 39.6%	250.000

أكتب ونفذ برنامج يقرأ قيمة الدخل بالدولار ويطبع الضرائب المستحقة.

33.2 أكتب ونفذ برنامج يقرأ التقدير A أو B أو C أو D أو F ثم يطبع "excellent" أو "good" أو "fair" أو "poor" أو "failure" استخدام عبارة switch.

34.2 أكتب ونفذ برنامج يقرأ حرف ويستخدم جملة swich ليطبع "do" إذا كان الحرف هو C ويطبع "re" إذا كان الحرف هو D ويطبع "me" إذا كان الحرف هو E ويطبع "fa" إذا كان الحرف هو F ويطبع "sol" إذا كان الحرف هو G ويطبع "la" إذا كان الحرف هو A ويطبع "ti" إذا كان الحرف هو B ويطبع "error" لأي حرف آخر.

35.2 أكتب ونفذ برنامج يقرأ ويطبع "It is a vowel" إذا كان الحرف لين ويطبع "It is an operator" إذا كان الحرف واحد من المعاملات الحسابية الخمسة ويطبع "It is something else" لأي شئ آخـر . استخدم جملة swich.

36.2 أكتب ونفذ برنامج يقرأ عدد مفرد ويطبع هذا العدد بالحروف . على سبيل المثال لو أن العدد الداخل إلى البرنامج هو 7 فإن خرج البرنامج يكون الكلمة "seven" . استخدم swich.

37.2 أكتب ونفذ برنامج يقرأ حرفين وعددين صحيحين . إذا كان الحرف الأول أو الحرفين معاً يكونوا واحد من المعاملات العلاقية الستة عند ذلك يتم مقارنة العددين بهذا المعامل وتطبع رسالة بنتيجة المقارنة. على سبيل المثال تنفيذ البرنامج مثل الآتي:

```
! = 33   77
33  is not equal to  77
```

38.2 عدل البرنامج الذي في المثال 10.2 باستبدال ثاني if بـ else if وثالث if بـ else . ما تأثير هذا التعديل على كفاءة البرنامج؟ كم شرط يتم إختبارها في كل تنفيذ للبرنامج في المتوسط.

39.2 أكتب ونفذ برنامج يقرأ ثلاثة أعداد صحيحة ويطبع الأقل والأكبر من هذه الأعداد . استخدم معامل التعبير الشرطي.

40.2 عدل برنامج المسألة 18.2 بحيث يكون أكثر صداقة للمستخدم . استخدم char بدلاً من int لمتغير الدخل بحيث يسمح للمستخدم بإدخال الحرف "r" لـ "rock" والحرف "p" لـ "paper" والحرف "s" لـ "scissors" .

إجابات لأسئلة المراجعة

1.2
```
if (cout > 100)  cout << "too many";
```

2.2 الكلمة المحجوزة هي كلمة مفتاحية في لغة البرمجة تحدد تركيب الجملة. على سبيل المثال الكلمات المفتاحية if و else هي كلمات محجوزة . المميز القياسي standard identifier هو كلمة مفتاحية تعرف نوع البيانات. من بين 48 كلمة مفتاحية في لغة ++C ، if و else و while هي كلمات محجوزة و char و int و float هي مميزات قياسية.

3.2 التعبير القصري "short-circuiting" يستخدم لوصف الطريقة التي تستخدم في لغة ++C لتقدير التعبيرات المنطقية المركبة مثل (x > 2 || y > 5) و (x > 2 && y > 5) . إذا كانت x أكبر من 2 في التعبير الأول فإن قيمة y لا تقدر. في هذه الحالات الجزء الأول فقط من التعبير المركب هو الذي قدّر لأن قيمته تحدد القيمة الحقيقية للتعبير المركب.

4.2 هـذا التـعبير يـقدر بالقيمة 1- إذا كانت x < y ويقدر بالقيـمة 0 إذا كانت x == y ويقدر بالقيمة 1 إذا كانت x > y.

5.2 الإخفاق "fall through" في جملة switch هي حالة لا تحتوي علي جملة break لذلك تسبب استمرار سريان البرنامج في الطريق الصحيح إلى أوامر الحالة التالية.

6.2 !(p || q) ليست مثل !p || !q على سبيل المثال إذا كانت p صحيح و q غير صحيح فإن التعبير الأول يكـون غيـر صحيـح ولـكن التعبير الثاني يكون صحيح . التعبير الصحيح المكافئ للتعبير !(p || q) هو !p && !q.

b. !!!p هي نفس !p

c.

P && q || r ليست هي نفسها p && (q || r) على سبيل المثال إذا كانت p غير صحيح و r صحيح فإن التعبير الأول يكون صحيح ولكن التعبير الثاني يكون غير صحيح :

p && q || r هو نفس (p && q) || r.

7.2 التعريف الثاني enum هو إعادة تعريف الثوابت spring و summer و fall .

8.2 البنود enumerators يجب أن تكون مميزات صحيحة. سلسلة الحروف مثل "Tom" و "Dick" ليست مميزات .

9.2 المبرمج من المحتمل كان يقصد إختبار الشرط (x == 0) . ولكن باستخدام معامل التخصيص "=" بدلاً من معامل التساوي "==" فإن النتيجة سوف تختلف جذرياً عن ما كان يقصده المبرمج . على سبيل المثال إذا كانت قيمة x هي 22 قبل جملة if فإن جملة if سوف تغير قيمة x إلى صفر. علاوة على ذلك فإن التعبير (x = 0) سوف تقدر قيمته بصفر وهذا يعني أن هذا التعبير غير صحيح لذلك فإن جزء else الشرطي سوف ينفذ مقرراً أن قيمة x ليست صفراً.

10.2 من المحتمل أن المبرمج كان يقصد اختبار الشرط (x < y && y < z) . البرنامج كما هو مكتوب سوف يترجم وينفذ ولكن ليس كما كان مقصود . على سبيل المثال إذا كانت قيم x و y و z السابقة هي 44 و 66 و 22 بالترتيب فإن الشرط الجبري "x < y < z" يكون غير صحيح. ولكن كما هو مكتوب فإن هذا الجزء من البرنامج سوف يقدر من اليسار إلى اليمين كالآتي (x < y) < z . الشـرط الأول x < y سوف يكون حقيقي . ولكن قيمته العددية هي 1 لذلك فإن التعبير (x < y) سوف تقدر قيمته بـ 1 . عند ذلك فإن التعبير المركب (x < y) < z سوف يقدر كالآتي (1) < 66 وهذا أيضاً صحيح. لذلك فإن جملة الخرج سوف تنفذ مسجلة الرسالة الخاطئة 22 > 66 > 44 .

11.2

a.

إما cout يجب أن تستخدم بدلاً من cin أو معامل الإدخال << يجب أن يستخدم بدلاً من معامل الإخراج >> .

b.

يجب وجود الأقواس حول الشرط x < y والفاصلة المنقوطة لابد من وجودها في نهاية جملة if قبل كلمة else .

12.2 يوجد أكثر من جملة بين عبارة if وعبارة else . وهذه الجمل تحتاج أن تكون في جملة مركبة وذلك بوضعهم بين القوسين { } .

الفصل الثالث

3

أنواع التكرار والأعداد الحقيقية

Iteration and Floating Types

المقصود بالتكرار هو تكرار تنفيذ جملة أو مجموعة من الجمل في البرنامج . لغة ++C تحتوي على ثلاثة عبارات لتنفيذ عملية التكرار: عبارة while و while .... do و for . جمل أو عبارات التكرار تسمى الحلقات التكرارية loops.

1.3 الحلقة التكرارية while

الحلقة التكرارية while تكون في الصورة التالية :

```
while (condition) statement;
```

قيمة الشرط condition تقدر أولاً . إذا كانت هذه القيمة ليست صفراً (أي صحيح أو حقيقي) يتم تنفيذ الأمر statement ثم يعاد تقدير قيمة الشرط مرة أخرى. هاتين الخطوتين يتم تكرارهما إلى أن تكون قيمة الشرط صفراً (أي أنها غير صحيح) . لاحظ ضرورة وجود الأقواس حول الشرط.

مثال 1.3 طباعة مكعبات الأعداد

هذا البرنامج يستخدم الحلقة التكرارية while لطباعة مكعبات الأعداد :

```
main ()
{
    int n;
    cout << "Enter positive integers.    Terminate with 0. \n\t: ";
    cin >> n;
    while (n > 0) {
```

```
        cout << n << " cubed is " << n*n*n* << ";
        cin >> n;
    }
}
```

```
Enter positive integers. Terminate with 0.
        : 2
2 cubed is 8
        : 5
5 cubed is 125
        : 7
7 cubed is 343
        : 0
```

أول قيمة أدخلت للمتغير n كانت 2. الحلقة while تختبر الشرط (n > 0) . حيث أن هذا الشرط صحيح فإن الجملتين الموجودتين داخل الحلقة التكرارية (Loop) يتم تنفيذهم . الجملة الثانية في داخل الحلقة التكرارية تقرأ العدد 5 للمتغير n . حيث أن الشرط (n > 0) مازال صحيح فإن الجملتين داخل الحلقة التكرارية يتم تنفيذهم مرة أخرى. في كل مرة يتم إختبار الشرط مرة أخرى . بعد نهاية التكرار الثالث يأخذ المتغير n القيمة صفر ويصبح الشرط غير صحيح . وبهذا ينتهي تكرار تنفيذ الحلقة.

يترك معظم المبرمجين بلغة ++C مسافة في بداية كل الجمل التي تقع داخل الحلقة التكرارية.

مثال 2.3 مجموع مربعات الأعداد

هذا البرنامج يستخدم الحلقة التكرارية while لإيجاد مجموع الأعداد الصحيحة من 1 إلى n :

$$\sum_{i=1}^{n} i^2 = 1^2 + 2^2 + 3^2 + ... + n^2$$

```
main ()
{
    int i = 1, n, sum = 0;
    cout << " Enter a positive integer: ";  cin >> n;
```

```
    while (i <= n) {
        sum += i*i;
        i++;
    }
    cout << "The sum of the first " << n << " squares is "
        << sum << endl;
}
```

```
Enter a positive integer: 4
The sum of the frist 4 squares is 30
```

```
Enter a positive integer : 6
The sum of the first 6 squares is 91
```

أول تنفيذ للبرنامج يحسب مجموع مربعات الأربعة أعداد الأولى : 1+4+9+16 = 30. ثاني تنفيذ للبرنامج يحسب مجموع مربعات الستة أعداد الأولى : 1+4+9+16+25+36 = 91 . عندما ترغب في تنفيذ مجموعة جمل داخل الحلقة التكرارية يجب أن تجمع هذه الجملة في جملة مركبة وذلك باستعمال الأقواس { }. مثال 2.3 يوضح الطريقة القياسية لشكل الجملة المركبة داخل الحلقة التكرارية. القوس الأيسر في نهاية السطر العلوي للحلقة التكرارية، والقوس اليمين في سطر مستقل تحت حرف "w" من الكلمة المفتاحية while. والمقصود تنفيذ كل الجمل الموجودة في الجملة المركبة بين القوسين { } .

بالطبع فإن المترجم لا يبالي بشكل هذا الجزء من البرنامج . فإنه يقبل هذه الصورة :

```
while (i <= n)  { sum += i*i; i++; }
```

ولكن معظم المبرمجين بلغة ++C وجدوا أن الصورة الظاهرة أسهل في القراءة . بعض المبرمجين بلغة C يفضلوا وضع القوس الأيسر في سطر مستقل تحت حرف "w" من كلمة while.

2.3 الحلقة do ... while

الحلقة التكرارية do ... while هي غالباً مثل الحلقة while . وتكون في الصورة التالية :

```
do statement while (condition);
```

الفرق الوحيد هو أن جملة الحلقة التكرارية do ... while تبدأ بتنفيذ العملية statmenet أولاً وتنتهي باختبار الشرط condition . هاتين الخطوتين يتم تكرارهما إلى أن تكون قيمة الشرط صفراً (أي غير صحيح) . الحلقة التكرارية do ... while دائماً تنفذ مرة واحدة على الأقل بغض النظر عن قيمة الشرط condition لأن العملية تنفذ في المرة الأولى قبل تقدير قيمة الشرط .

مثال 3.3 دالة المضروب

هذا البرنامج يحسب دالة المضروب : $n! = (n)(n-1) \ldots (3)(2)(1)$.

```
main ( )
{
    int  n,  f = 1;
    cout  <<  "Enter a positive integer: ";    cin  >>  n;
    cout  <<  n  <<  " factorial is " ;
    do  {
        f  *=  n;
        n--;
    } while  (n  >  1);
    cout  <<  f  <<  endl;
}
```

```
Enter a positive integer:  5
5  factorial is 120

Enter a positive integer: 8
8  factorial is 40320
```

البرنامج يخصص قيمة ابتدائية للمتغير f وبعد ذلك يضرب هذه القيمة في قيمة المتغير n وكل الأعداد الصحيحة الموجبة التي أقل من قيمة n . لذلك $5! = (5)(4)(3)(2)(1) = 120$. ومضروب العدد 8 هو $8! = (8)(7)(6)(5)(4)(3)(2)(1) = 40320$.

3.3 الحلقة التكرارية for

يتم التحكم في هذه الحلقة التكرارية بثلاثة أجزاء منفصلة : القيمة الابتدائية initialization وشرط الاستمرار continuation condition والقيمة الجديدة update . على سبيل المثال في البرنامج الذي في المثال 3.3 المتحكم في الحلقة التكرارية هو المتغير n قيمته الابتدائية هي cin >> n، شرط الاستمرار هو n > 1، والقيمة الجديدة هي -- n . عندما تكون هذه الأجزاء الثلاثة مبسطة فإن الحلقة التكرارية يمكن أن توضع في شكل الحلقة التكرارية for التي هي عادة أبسط من نظيرتها while و do ... while . الصورة العامة لجملة الحلقة التكرارية for هي :

for (*initialization; continuation condition; update*) *statement;*

القيمة الابتدئية وشرط الاستمرار والقيمة الجديدة يمكن أن يكونوا فارغين أي بدون أي قيم.

مثال 4.3 مجموع مربعات الاعداد مرة اخرى

هذا البرنامج له نفس الأداء مثل البرنامج الذي في المثال 2.3 :

```
main ()
{
    int n, sum = 0;
    cout << "Enter a positive integer: ";
    cin >> n;
    for (int i = 1; i <= n; i++)
        sum += i*i;
    cout << "The sum of the first " << n << " squares is "
        << sum << endl;
}
```

في هذا البرنامج القيمة الابتدائية هي int i = 1 وشرط الاستمرار هو i <= n والقيمة الجديدة هي ++i. من المعتاد أن يكون الإعلان عن متغير التحكم مع تخصيص القيمة الابتدائية له في الحلقة التكرارية for . على سبيل المثال متغير التحكم i في البرنامج السابق تم الإعلان عنه بداخل جزء التخصيص int i =1 . وهذه خاصية جميلة في لغة ++C . على أي حال بمجرد الإعلان عن متغير التحكم بهذه الطريقة فإنه يجب أن لا يعاد الإعلان عنه في حلقة تكرارية for فيما بعد . على سبيل المثال

```
for (int i = 0; i < 100; i++)
    sum += i*i;
```

```
for (int i = 0; i < 100; ; i++)  // ERROR: i has already been declared
    cout << i*i*i;
```

نفس متغير التحكم يمكن أن يستخدم مرة أخرى بدون إعادة الإعلان عنه مرة أخرى:

```
for (i = 0;  i < 100; i++)                    // OK
    cout << i*i*i;
```

إذا كان عندك حرية الإختيار بين الحلقات التكرارية for و while و do ... while من المحتمل أنك سوف تستخدم الحلقة التكرارية for . كما يوضح المثال التالي ، سنجد الحلقة التكرارية for سهلة الفهم.

مثال 5.3 دالة المضروب مرة أخرى

قارن هذا البرنامج مع البرنامج الذي في المثال 3.3

```
main ()
{
    int n, f = 1;
    cout << "Enter a positive integer: ";    cin >> n;
    for (int i = 2; i <= n; i++)
        f *= i;
    cout << n << " factorial is " << f << endl;
}
```

هذا البرنامج يحسب المضروب بضرب العدد 1 في الأعداد 2 و 3 و ..... و n-1 و n . تنفيذ هذا البرنامج ليس أسرع من البرنامج الذي يستخدم الحلقة التكرارية while ولكن هذا البرنامج أكثر فهماً.

مثال 6.3 القيم العظمى والصغرى في متتالية عددية

هذا البرنامج يقرأ متتالية من الأعداد الصحيحة الموجبة تنتهي بالعدد الصحيح صفر بعد ذلك يطبع أصغر وأكبر الأعداد في المتتالية .

```
main ()
{
    int n, min, max;
    cout << "Enter positive integers.  Terminate input with 0: \n" '
    cin >> n;
```

```
    for (min = max = n;  n > 0;) {
        if (n < min) min = n;           // min and max are the smallest
        else if (n > max) max = n;      // and largest of the n that
        cin >> n;                       // have been read so far
    }
    cout << "min = " << min << " and max =" << max << endl;
}
```

```
Enter positive integers.  Terminate input with 0:
55
22
88
66
0
min = 22 and max = 88
```

لاحظ أن جزء التخصيص الابتدائي في الحلقة التكرارية for هو min = max = n يكافئ عملية تخصيص لمتغيرين وجزء القيمة الجديدة فارغ أو محذوف . لاحظ أيضاً استعمال التعليق الذي يشمل ثلاثة سطور في البرنامج . هذا التعليق يصف الحلقة التكرارية وشرط المتغيرات يجب أن تكون صحيحة في كل تنفيذ للحلقة التكرارية.

الحارس sentinel هي قيمة خاصة للمتغير تستخدم لإنهاء الحلقة التكرارية الداخلية . في المثال السابق القيمة صفر استخدمت كحارس.

مثال 7.3 أكثر من متغير تحكم

هذا البرنامج يبين كيفية استخدام أكثر من متغير تحكم في الحلقة التكرارية for .

```
main ()
{
    for (int m = 1, n = 8; m < n; m++, n--)
        cout << "m = " << m << ", n = " << n << endl;
}
```

```
m = 1,  n = 8
m = 2,  n = 7
m = 3,  n = 6
m = 4,  n = 5
```

جزء التخصيص الابتدائي في الحلقة التكرارية for يعلن عن متغيرين تحكم هما m و n التخصيص الابتدائي للمتغير m هو 1 وللمتغير n هو 8 . جزء القيمة الجديدة يستخدم الفاصلة بين تعبيرات القيم الجديدة: ++m و --m . الحلقة التكرارية تستمر مادام m < n . (لاحظ أن الفاصلة في جزء التخصيص الابتدائي تستخدم كجزء من قائمة التخصيص الابتدائي) .

4.3 الأمر Break

لقد رأينا سابقاً استعمال الأمر Break في جملة switch ، إنها تستخدم أيضاً في الحلقات التكرارية. عند تنفيذ الأمر Break فإنه ينهي الحلقة التكرارية ويخرج من التكرار عند هذه النقطة.

مثال 8.3 الخروج من حلقة لا نهائية

هذه الحلقة التكرارية while تكافئ الحلقة الموجودة في المثال 2.3 :

```
while (1) {
    if (i > n) break;              // loop stops here when i > n
        sum += i*i;
        i++;
}
```

مادام الشرط (i <= n) صحيحاً فإن الحلقة التكرارية سوف تستمر كما في المثال 2.3 . ولكن بمجرد أن تكون (i > n) فإن الأمر Break ينفذ وينهي الحلقة التكرارية فوراً.

مثال 9.3 التحكم في الدخل عن طريق الرقم صفر

هذا البرنامج يقرأ متتالية من الأعداد الصحيحة الموجبة تنتهي بالصفر ويطبع متوسط هذه الأعداد :

```
main ()
{
    int n, count = 0, sum = 0;
    cout << "Enter positive integers.  Terminate input with 0: \n";
    for ( ; ; ) {
        cout << "\t" << count + 1 << " : ";
```

```
        cin >> n;
        if (n == 0) break;
        ++count;
        sum += n;
    }
    cout << "The average of the " << count << " numbers is "
         << float (sum) / count << endl;
}
```

```
Enter positive integers.   Terminate input with 0:
        1 : 7
        2 : 4
        3 : 5
        4 : 2
        5 : 0
The average of the 4 numbers is 4.5
```

عند إدخال العدد صفر فإن الأمر break ينفذ وينهي مباشرة الحلقة التكرايرة for حيث تنفيذ أخر جملة في الخرج.

لاحظ أن كل الثلاثة أجزاء الخاصة بالتحكم في الحلقة التكرارية for فارغة أو محذوفة : (; ;) for . هذا التركيب لـ for يطلق عليه "إلى ما لا نهاية" "forever" . بدون وجود الأمر Break فإنها تكون حلقة تكرارية لا نهائية.

5.3 الأمر Continue

الأمر break يجعلنا نقفز على كل الجمل المتبقية في بلوك الحلقة التكرارية ونخرج من الحلقة إلى الجملة التالية بعد الحلقة . جملة continue تفعل نفس الشئ ماعدا أنه بدلا من الخروج من الحلقة فإنها تعود مرة أخرى إلى بداية الحلقة لتبدأ التكرار التالي للحلقة.

مثال 10.3 إستخدام الأوامر continue و break

هذا البرنامج الصغير يوضح إستعمال الأمرين continue و break :

```
main ()
{
    int n;
    for ( ; ; ) {
        cout << "Enter int: ";   cin >> n;
        if (n%2 == 0) continue;
        if (n%3 == 0) break;
        cout << "\tBottom of loop. \n";
    }
    cout << "\tOutside of loop. \n";
}
```

```
Enter int : 7
        Bottom of loop.
Enter int : 4
Enter int : 9
        Outside of loop.
```

عندما تكون قيمة المتغير n تساوي 7 فإن كل من الشرطين يكون غير صحيح وسريان البرنامج يصل إلى نهاية الحلقة. عندما تكون قيمة n تساوي 4 فإن الشرط الأول يكون صحيح (4 ضعف 2) لذلك فإن سريان البرنامج يتخطى الجمل الباقية في الحلقة ويقفز مباشرة إلى أعلى الحلقة مرة أخرى ليستمر في تنفيذ التكرار التالي. عندما تكون قيمة n تساوي 9 فإن أول شرط سوف يكون غير صحيح (9 ليست ضعف 2) ولكن الشرط الثاني سوف يكون صحيح (9 ضعف 3) لذلك فإن سريان البرنامج سوف يخرج من الحلقة مباشرة إلى أول جملة بعد الحلقة .

6.3 الأمر goto

العبارات break و continue و swich تسبب تغير سريان تنفيذ البرنامج عن مساره الطبيعي. المكان المقصود الذهاب إليه يتحدد بالجمل : break للذهاب إلى الجملة التالية خارج الحلقة و continue للذهاب إلى شرط استمرارية الحلقة و swich للذهاب إلى الحالة الصحيحة . هذه الأوامر الثلاثة تسمى جمل القفز jump statements لأنها تسبب تخطي البرنامج لبعض الجمل.

الأمر goto هو نوع آخر من أوامر القفز . المكان المراد الذاهاب إليه يحدد داخل الأمر بكلمة تدل عليه ، وتسمى الدليل label.

الدليل هو ببساطة مميز مسبوق بعلامة الترقيم ":" في بداية الجملة.

هذه الأدلة تعمل مثل عبارات case داخل جملة switch : تحدد المكان المراد الذهاب إليه.

مثال 11.3 الخروج من الحلقات المتداخلة

هذا البرنامج يوضح الطريقة الصحيحة للخروج من الحلقات المتداخلة :

```
main ()
{
     int a, b, c;
     cin >> a >> b >> c;
     for (int i = 0; i < a; i++) {
          for (int j = 0; j < b; j++)
               for (int k = 0; k < c; k++)
                    if (i*j*k > 100) goto esc;
                    else cout << i*j*k << " ";
esc: cout << endl;
     }
}
```

عند الوصول إلى جملة goto في الحلقة الداخلية فإن البرنامج يقفز إلى الخارج لتنفيذ جملة الخرج التي أسفل الحلقة الخارجية.

يوجد طرق أخرى للخروج من الحلقات المتداخلة . أحد هذه الطرق هي خمد متغير التحكم في الحلقة باستبدال جملة if داخل الحلقة k كالتالي :

```
if (i*j*k > 100) j = k = b + c;
else cout << i*j*k << " ";
```

هذا سوف يسبب إنهاء الحلقة j والحلقة k لأن شروط استمرارهم j < b و k < c سوف يكون غير صحيح. هذه هي طريقة القرصنة "hacker's method" لأنها تضع قيم زائفة لمتغيرات التحكم j و k للخروج من الحلقة.

طريقة أخرى هي استخدام اشارة "done flag" داخل شروط الاستمرارية للحلقات التكرارية for كالتالي:

```
int done = 0;
for (int i = 0; i < a && !done; i++) {
    for (int j = 0; j < b && !done; j++)
        for (int k = 0; k < c && !done; k++)
            if (i*J*K > 100) done = 1;
            else cout << i*j*k << " ";
}
```

تعتبر هذه أيضاً طريقة شاقة . إستخدام عبارة goto هي أفضل طريقة لإنهاء الحلقات المتداخلة .

المثال التالي يوضح إستعمال الأمر goto :

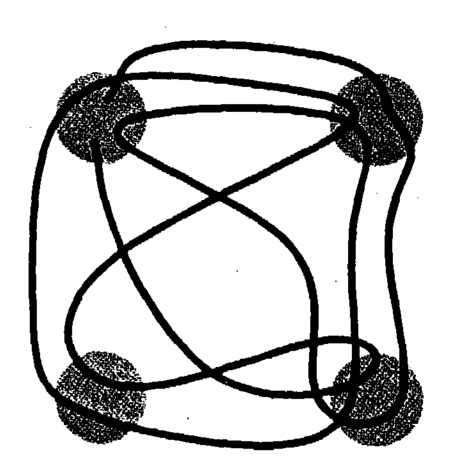
مثال 12.3 الإفراط في استخدام جمل goto

هذا البرنامج يبين كيف أن استخدام الأمر goto يمكن أن يؤدي إلى برنامج متشابك "spaghetti code" :

```
main ()
{
    int n;
    cout << "Enter n: ";
    cin >> n;
s1: cout << "Now at step 1 with n = " << n << endl;
    --n;
    if (n < 2) return 0;
s2 : cout << "Now at step 2 with n = " << n << endl;
    if (n < 7) goto s4;
s3 : cout << "Now at step 3 with n = " << n << endl;
    if (n % 2 == 0) goto s1;
s4 : cout << "Now at step 4 with n = " << n << endl;
    n -= 2;
    if (n > 4) goto s1;
    else goto s3;
}
```

```
Enter n : 9
Now at step 1 with n = 9
Now at step 2 with n = 8
Now at step 3 with n = 8
Now at step 1 with n = 8
Now at step 2 with n = 7
Now at step 3 with n = 7
Now at step 4 with n = 7
Now at step 1 with n = 5
Now at step 2 with n = 4
Now at step 4 with n = 4
Now at step 3 with n = 2
Now at step 1 with n = 2
```

حيث أن المتغير n يتناقص من 9 إلى 2 فإن جمل goto تغير مسار البرنامج إلى الخلف وإلى الأمام خلال جمل الخرج الأربعة المحددة بالأدلة s1 و s2 و s3 و s4 .



إستعمال goto بطريقة غير محكمة قد يؤدي إلى برنامج متشابك يصعب تصحيح أخطاؤه.

7.3 أنواع الأعداد الحقيقية

لغة ++C تحتوي على ثلاثة أنواع للأعداد الحقيقية : float و double و long double . على معظم أجهزة الحاسب النوع double يشغل حيز من البايت ضعف عدد البايت التي يشغلها النوع float . بالتحديد النوع float يشغل 4 بايت والنوع double يشغل 8 بايت والنوع long double يشغل 8 أو 10 أو 12 أو 16 بايت.

الأنواع التي تسخدم في الأعداد الحقيقية تسمى أنواع العلامة العشرية "floating point" وذلك لطريقة تخزين هذه الأنواع داخل الحاسب . على معظم أجهزة الحاسب رقم مثل 123.45 يحول أولاً إلى الصورة الثنائية binary form :

$$123.45 = 1111011.01110011_2$$

بعد ذلك تتحرك النقطة العشرية بحيث أن كل البتات تكون على يمينها. في هذا المثال صيغة العلامة العشرية المتحركة نحصل عليها بتحريك العلامة العشرية 7 بتات إلى اليسار وبذلك يكون الجزء العشري mantissa أقل من 2^7 . لذلك فإن العدد الأصلي يكون

$$123.45 = 0.111101101110011_2 \times 2^7$$

هذا العدد سوف يمثل داخلياً بتخزين الجزء العشري 0.111101101110011 والجزء الأسي منفصلين. في نوع float الذي يتكون من 32 خانة يخزن الجزء العشري في 23 بت والجزء الأسي في 8 بتات وخانة لإشارة العدد. في النوع double الذي يتكون من 64 بت يخزن الجزء العشري في 52 بت والجزء الأسي في 11 بت.

المثال التالي يمكن أن يستخدم على أي جهاز حاسب لتحديد عدد البايتات التي يستخدمها لكل نوع. البرنامج يستخدم المعامل sizeof الذي يحدد الحجم المعين لكل نوع بالبايت.

مثال 13.3 استخدام المعامل sizeof

هذا البرنامج يخبرك بالحيز الذي يشغله كل نوع من أنواع الأعداد الإثنى عشر المستعملة:

```
main () {
    cout << "Number of bytes used: \n";
    cout << "\t           char :  " <<          sizeof (char)   << endl;
    cout << "\t          short :  " <<         sizeof (short)   << endl;
    cout << "\t            int :  " <<           sizeof (int)   << endl;
    cout << "\t           long :  " <<          sizeof (long)   << endl;
    cout << "\t unsigned char :   " <<  sizeof (unsigned char)  << endl;
    cout << "\tunsigned short :   " << sizeof (unsigned short)  << endl;
    cout << "\t  unsigned int :   " <<   sizeof (unsigned int)  << endl;
    cout << "\tunsigned long :    " <<  sizeof (unsigned long)  << endl;
    cout << "\t  signed char :    " <<    sizeof (signed char)  << endl;
    cout << "\t          float :  " <<          sizeof  (float) << endl;
    cout << "\t        double :   " <<         sizeof (double)  << endl;
    cout << "\t  long double :    " <<   sizeof ( long double)  << endl;
    }
```

خرج هذا البرنامج يبين الحجم بالبايت الذي يشغله كل نوع من الأنواع الموجودة على محطة تشغيل UNIX . على هذا الحاسب النوع int و long متكافئين والنوع unsigned int و unsigned long متكافئين والنوع double و long double متكافئين . بمعنى آخر أنه لا يوجد فرق بين "long" و "regular" على هذا الحاسب.

```
Number of bytes used:
                 char : 1
                short : 2
                  int : 4
                 long : 4
        unsigned char : 1
       unsigned short : 2
         unsigned int : 4
        unsigned long : 4
          signed char : 1
                float : 4
               double : 8
          long double : 8
```

البرنامج التالي يمكن أن يستخدم في البحث عن أنواع الأعداد الحقيقية على أي جهاز حاسب . البرنامج يقرأ قيم لثوابت متعددة من الملف الرأسي float.h . للوصول إلي هذا الملف يجب أن يتضمن البرنامج توجيه المعالج الأولي:

```
# include <float.h>
```

هذا مثل التوجيه <iostream.h>#include الذي يكون موجود دائماً لاستعمال الأهداف cin و cout .

مثال 14.3 القراءة من الملف float.h

هذا البرنامج يخبرك بمدى المقدار والدقة في النوع float على جهازك.

```
main () {
    int fbits = 8 * sizeof (float);              // each byte contains 8 bits
    cout << "float uses: \t" << fbits    <<  " bits : \n\t\t"
         <<    FLT_MANT_DIG - 1  <<  " bits for its mantissa \n\t\t "
         << fbits - FLT_MANT_DIG  <<  " bits for its exponent \n\t\t "
```

```
        <<                              1  <<    " bit for its sign \n"
        << "               to obtain : "    << FLT_DIG << " sig. digits \n"
        << "    with minimum value  : "    << FLT_MIN << endl
        << "     and maximum value  : "    << FLT_MAX << endl;
}
```

```
float uses :            32  bits :
                        23  bits for its mantissa
                         8  bits for its exponent
                         1  bit for its sign
          to obtain :  6 sig. digits
with minimum value :  1.17549e-38
 and maximum value :  3.40282e+38
```

الثوابت FLT_MANT_DIG و FLT_DIG و FLT_MIN و FLT_MAX معرفة في الملف الرأسي float.h .

خرج هذا البرنامج هو من محطة تشغيل UNIX . فهو يبين أن 32 بت التي يستخدمها لتخزين النوع float مقسمة إلى ثلاثة أجزاء : 23 بت للجزء العشري و 8 بت للجزء الأسي وبت واحد للإشارة. الـ 23 بت الخاصة بالجزء العشري تنتج قيمة تتكون من 6 خانات رقمية والـ 8 بت الخاصة بالأس تنتج قيمة في المدى من 10^{-37} إلى 3×10^{38} .

كل حسابات الأعداد الحقيقية تتم بدقة مضاعفة double . لذلك فإن الوقت الوحيد الذي يجب أن تستخدم فيه float بدلاً من double هو عند تخزين كميات كبيرة من الأعداد الحقيقية آخذاً في الإعتبار حيز الذاكرة أو سرعة الوصول للمعلومة.

8.3 تحويلات النوع

رأينا في الفصل الثاني كيف أن أي نوع من أنواع الأعداد الصحيحة يمكن أن يتحول تلقائياً إلي نوع آخر للأعداد الصحيحة . أيضاً في لغة ++C يمكن تحويل أنواع الأعداد الصحيحة إلى أنواع الأعداد الحقيقية تلقائياً. على سبيل المثال :

```
int n = 22;
float x = 3.14159;
x += n;                          // the value 22 is automatically converted to 22.0
cout << x - 2 << endl;           // value 2 is automatically converted to 2.0
```

في هذا المثال تم تحويل العدد الصحيح 22 إلى عدد حقيقي 22.0 تلقائياً. ولكن التحويل من عدد حقيقي إلى عدد صحيح لا يتم تلقائياً .

بصفة عامة إذا كان T هو أحد الأنواع و v هو قيمة النوع الآخر فإن التعبير

```
T (v)
```

يحول القيمة v إلى النوع T . وهذا يسمى تحويل النوع type casting . على سبيل المثال إذا كان expr هو تعبير من نوع الأعداد الحقيقية و n هو متغير من نوع الأعداد الصحيحة int فإن الجملة الآتية

```
n = int (expr);
```

تحول قيمة expr إلي نوع الأعداد الصحيحة int ويخصصها للمتغير n . على سبيل المثال 2.71828 سوف يحول إلى 2 . لاحظ أن هذا هو حذف الرقم العشري وليس تقريب له.

مثال 15.3 مثال بسيط لتحويل النوع

هذا البرنامج يحول النوع double إلى النوع int

```
main ()
{
    double v = 1234.56789;
    int n = int (v);
    cout << "v = " << v << ", n = " << n << endl;
}
```

```
v = 1234.57 , n = 1234
```

القيمة 1234.56789 التي من النوع الحقيقي double حولت إلى القيمة 1234 التي من النوع الصحيح int . عندما يتم التحويل من نوع إلى نوع آخر أعلى فإن معامل التحويل لا ضرورة له . نحن ذكرنا في الباب الثاني أن هذا يسمى ترقية النوع type promotion . مثال آخر لترقية النوع من النوع char إلى النوع double .

مثال 16.3 انواع الترقي

هذا البرنامج يرقي النوع char إلى short

```
main ()
{
    char c = 'A';         cout << "  char c =  " << c << endl;
    short k = c;          cout << "  short k =  " << k << endl;
    int m = k;            cout << "   int m =  " << m << endl;
    long n = m;           cout << "  long n =  " << n << endl;
    float x = n;          cout << "  float x =  " << x << endl;
    double y = x;         cout << " double y =  " << y << endl;
}
```

```
  char c =  A
 short k =  65
   int m =  65
  long n =  65
  float x =  65
double y =  65
```

القيمة الصحيحة للحرف 'A' هي الأسكي كود 65 . هذه القيمة الصحيحة تخزن في المتغير c في صورة char كما تخزن في المتغير k في صورة short كما تخزن في المتغير m في صورة int كما تخزن في المتغير n في صورة long . بعد ذلك تحول هذه القيمة إلى عدد حقيقي هو 65.0 وتخزن في المتغير x في صورة float وفي المتغير y في صورة double . لاحظ أن cout يطبع العدد الصحيح c كحرف ويطبع الأعداد الحقيقية x و y في صورة أعداد صحيحة لأن الجزء العشري يساوي صفر.

حيث أن التحويل بين الأعداد الحقيقية والأعداد الصحيحة سهل جداً في لغة ++C فإنه من السهل نسيان التميز بينهما . عموماً الأعداد الحقيقية تستخدم لحساب الأشياء العددية التي ليس بها كسور بينما الأعداد الحقيقية تستخدم في قياس الأشياء التي يمكن أن تحتوي على كسور . هذا يعني أن الأعداد الصحيحة هي قيم مضبوطة بينما الأعداد الحقيقية هي قيم تقريبية .

9.3 الخطأ نتيجة تقريب الأعداد

في الحاسب ، أبسط قيم للأعداد الحقيقية تكون غير دقيقة مائة بالمائة. عدم الدقة هذه تسمى الخطأ نتيجة التقريب roundoff error .

مثال 17.3 الخطأ نتيجة التقريب

هذا البرنامج يقوم بعمل عمليات حسابية بسيطة لتوضيح الخطأ نتيجة التقريب :

```
main ()
{
    double x = 1000/3.0;        cout << "x = " << x << endl;
    double y = x - 333.0;       cout << "y = " << y << endl;
    double z = 3 * y - 1.0;     cout << "z = " << z << endl;
    if (z == 0)  cout << "z == 0. \n";
    else cout << "z does not equal 0. \n";
}
```

```
x = 333.333
y = 0.333333
z = -5.68434e-14
z does not equal 0.
```

في العمليات الحسابية المضبوطة قيم المتغيرات x = 333 1/3 و y = 1/3 و z = 0 لكن 1/3 لا يمكن تمثيله بالضبط كقيمة حقيقية تحتوي على كسر عشري . هذا ينعكس في القيمة المتبقية للمتغير z .

هذا المثال يوضح أيضاً المشكلة المتأصلة في استخدام الأعداد الحقيقية التي تحتوي على النقطة العشرية المتنقلة في اختبارات شروط التساوي. الاختبار (z == 0) سوف يفشل حتى إذا كانت قيمة z قريبة جداً من الصفر. لذلك من الأفضل تجنب اختبارات التساوي بأنواع الأعداد الحقيقية ذات النقطة العشرية المتنقلة.

10.3 الصيغة الأسية للأعداد الحقيقية

عندما يكون الدخل أو الخرج عبارة عن أعداد حقيقية تحتوي على النقطة العشرية فإنه يمكن وصفها بطريقتين : النقطة الثابتة fixed point والعلمية scientific . الخرج في المثال 17.3 يوضح كل منهما.

333.333 له صيغة النقطة الثابتة و 5.68434e-14- له الصيغة العلمية. في الصيغة العلمية الحرف e مخصص للأس 10 "exponent on 10" . لذلك 5.68434e-14- يعني 5.68434 x 10 -14- الذي يساوي 0.0000000000000568430- . واضح أن الصيغة العلمية أكثر كفاءة في الأرقام الصغيرة جداً والكبيرة جداً.

الأعداد الحقيقية التي قيمتها تنحصر بين 0.1 و 999999 سوف تطبع في صيغة النقطة الثابتة ، وكل القيم الأخرى سوف تطبع في الصيغة العلمية.

مثال 18.3 الصيغة العلمية

هذا البرنامج يبين كيفية إدخال الأعداد الحقيقية إلى الحاسب في الصيغة العلمية :

```
# include  <iostream.h>
main  ( )
{
      double  x;
      cout  <<  "Enter float : ";    cin  >>  x;
      cout  <<  "Its reciprocal is : "  <<  1/x  <<  endl;
}
```

```
Enter float :  234.567e89
Its reciprocal is :  4.26317e-92
```

تستطيع أن تستخدم إما e أو E في الصيغة العلمية .

11.3 الثوابت والمتغيرات والأهداف

الهدف object هو منطقة في الذاكرة لها عنوان وحجم ونوع وقيمة . عنوان الهدف هو عنوان الذاكرة في أول بايت. حجم الهدف هو عدد البايتات التي يشغلها في الذاكرة . قيمة الهدف هو ثابت محدد بواسطة البتات الفعلية المخزنة في الذاكرة وبنوع الهدف الذي يصف كيفية تفسير هذه البتات.

على سبيل المثال مع ++GNU C على محطة التشغيل UNIX الهدف n المعرف بالجملة الآتية:

```
int  n  =  22;
```

له عنوان في الذاكرة 0xfffcd3 وحجمه 4 بايتات نوعه int وقيمته 22 . (عنوان بنظام الأعداد الستعشري 16 . إنظر الملحق G).

نوع الهدف يحدد بواسطة المبرمج . قيمة الهدف يمكن أيضاً أن تحدد بواسطة المبرمج أثناء ترجمة البرنامج أو أثناء تنفيذ البرنامج . حجم الهدف يحدد بواسطة المترجم . على سبيل المثال في GNU C++ النوع int له حجم 4 بينما في بورلاند C++ حجم النوع int هو 2 بايت . عنوان الهدف يحدد بواسطة نظام تشغيل الحاسب أثناء تنفيذ البرنامج.

بعض الأهداف ليس لها أسماء . وسوف نرى أمثلة للأهداف التي ليس لها أسماء في الفصل الرابع والخامس . المتغير هو هدف له اسم . الهدف المعرف سابقاً هو متغير له اسم 'n' .

الكلمة متغير "variable" تستخدم للدلالة على أن قيمة الهدف يمكن أن تتغير . الهدف الذي قيمته لا يمكن أن تتغير يسمى ثابت (constant) . الثوابت يتم الإعلان عنها بالكلمة المفتاحية const التي تسبق نوع الثابت كما يلي :

```
const int N = 22;
```

مثال 19.3 تحديد الثابت const

هذا البرنامج يوضح تعريف الثابت

```
main ()
{
    const char BEEP = '\b ';
    const int MAXINT = 214783647;
    const int N = MAXINT/2;
    const float KM_PER_MI = 1.60934;
    const double PI == 3.14159265358979323846;
}
```

الثوابت تعرف عادة للقيم مثل π التي سوف تستخدم أكثر من مرة في البرنامج بدون تغيير. من المعتاد استخدام الحروف الكبيرة في مميزات الثوابت وذلك للتفرقة بينها وبين أنواع المميزات الأخرى. المترجم الجيد سوف يستبدل كل رمز لثابت بقيمته العددية .

12.3 توليد الارقام الشبه عشوائية

أحد التطبيقات الهامة للحاسبات هو محاكاة أو تمثيل الأنظمة الفعلية. معظم الأبحاث المتقدمة تعتمد بشدة على هذه الطريقة في دراسة كيفية تشغيل الأنظمة بدون التفاعل الحقيقي مع الأنظمة مباشرة.

المحاكاة تتطلب توليد الأرقام العشوائية من الحاسب لنمذجة الأشياء الغير معلومة. بالطبع الحاسبات لا تستطيع توليد أرقام عشوائية بدقة لأن الحاسبات محددة deterministic : إذا أعطى الحاسب نفس الدخل فإنه سوف ينتج نفس الخرج. لكن من الممكن توليد أرقام تظهر كما لو كانت مولدة عشوائياً ، يعني أن الأرقام موزعة بانتظام داخل فترة معلومة وليس لها شكل مميز . مثل هذه الأرقام تسمى الأرقام الشبه عشوائية Pseudo-random numbers .

ملف الرأسي في لغة C القياسية <stdlib.h> يعرف الدالة () rand التي تولد الأعداد الصحيحة الشبه عشوائية في المدى من صفر إلى RAND_MAX الذي هو معرف في الملف <stdlib.h> . كل وقت يتم فيه استدعاء الدالة ( ) rand فإنها تولد أرقام صحيحة أخرى في هذه الفترة.

مثال 20.3 توليد الأرقام الشبه عشوائية

```
# include  <iostream.h>
# include  <stdlib.h>

main ()
{
    for (int i = 0; i < 8 ; i++)
        cout << rand ( ) << endl;
    cout << "RAND_MAX = " << RAND_MAX << endl;
}
```

```
1103527590
377401575
662824084
1147902781
2035015474
368800899
1508029952
486256185
RAND_MAX = 2147483647
```

```
1103527590
377401575
66284084
1147902781
2035015474
368800899
1508029952
486256185
RAND_MAX = 2147483647
```

في كل تنفيذ للبرنامج يتم توليد 8 أرقام عشوائية صحيحة موجبة موزعة بالتساوي في الفترة من صفر إلى RAND_MAX التي هي 2147483647 على هذا الحاسب. لسوء الحظ كل تنفيذ للبرنامج ينتج نفس الأرقام متتابعة . والسبب في ذلك هو أن الأرقام تم توليدها من نفس البذرة "seed" .

كل من الأرقام الشبه عشوائية تم توليدها بتطبيق دالة خاصة "number crunching" معرفة داخلياً . أول رقم شبه عشوائي تم توليده من متغير معرف داخلياً يسمى البذرة seed للتتابع . في البداية هذا المتغير يأخذ قيمة ابتدائية بواسطة الحاسب تكون نفس القيمة في كل تنفيذ للبرنامج. للتغلب على هذه المشكلة يمكن أن تستخدم الدالة () srand لاختيار البذرة الخاصة بك.

مثال 21.3 توليد الارقام الشبه عشوائية

```
#include <iostream.h>
# include <stdlib.h>

main ()
{
     unsigned seed;
     cout << "Enter seed: " ;
     cin >> seed;
     srand (seed) ;                         // initializes the seed
     for (int i = o; i < 8; i++)
          cout << rand () << endl;
}
```

```
Enter seed: 0
12345
1406932606
654583775
1449466924
229283573
1109335178
1051550459
1293799192
```

```
Enter seed : 1
1103527590
377401575
662824084
1147902781
2035015474
368800899
1508029952
486256185
```

```
Enter seed : 12345
1406932606
654583775
1449466924
229283573
1109335178
1051550459
1293799192
794471793
```

السطر (srand (seed يخصص قيمة المتغير seed لـ seed الداخلية المستخدمة بالدالة () rand لتبدأ توليد الأرقام العشوائية المتتالية. إختلاف seeds يعطي نتائج مختلفة .

لاحظ أن قيمة البذرة 12345 المستخدمة في التنفيذ الثالث للبرنامج هي أول رقم تم توليده بالدالة () rand في أول تنفيذ للبرنامج . نتيجة لذلك فإن أول سبعة أرقام تم توليدها في التنفيذ الثالث للبرنامج هي نفس السبعة الأرقام الأخيرة المولدة في التنفيذ الأول للبرنامج . لاحظ أيضاً أن الأرقام المولدة في التنفيذ الثاني هي نفس الأرقام المولدة في المثال 20.3 . هذا يوحي أن قيمة seed الابتدائية لهذا الحاسب هي 1 .

مشكلة إدخال قيمة البذرة يمكن التغلب عليها باستخدام ساعة الحاسب. ساعة الحاسب system clock تراقب الوقت الحالي بالثانية. الدالة () time المعرفة في ملف الرأسي <time.h> تعطي الوقت الحالي كعدد صحيح بدون إشارة . وهذا يمكن أن يستخدم كبذرة في الدالة () rand.

مثال 22.3 توليد الارقام الشبه عشوائية

```
# include <iostream.h>
# include <stdlib.h>
# include <time.h>

main ()
{
    unsigned seed = time (NULL);          // uses the system clock
    cout << "seed = " << seed << endl;
    srand (seed) ;                        // initializes the seed
    for (int i = 0; i < 8; i++)
        cout << rand () << endl;
}
```

```
seed = 808148157
1877361330
352899587
1443923328
1857423289
200398846
1379699551
1622702508
715548277
```

```
seed = 808148160
892939769
1559273790
146864455
952730860
1322627253
1305580362
844657339
440402904
```

في أول تنفيذ للبرنامج تنتج الدالة () time الرقم الصحيح 808148157 الذي يستخدم كبذرة في مولد الأرقام العشوائية . التنفيذ الثاني للبرنامج يتم بعد 3 ثوان تأخير لذلك فإن الدالة () time تنتج العدد الصحيح 808148160 الذي يولد أرقام مختلفة تماماً في كثير من برامج المحاكاة يكون أحد المطالب هي توليد أرقام عشوائية تكون موزعة بإنتظام في فترة معينة . البرنامج التالي يوضح كيفية عمل ذلك.

مثال 23.3 توليد الارقام الشبه عشوائية

```
#include <iostream.h>
#include <stdlib.h>
#include <time.h>

main ()
{
    unsigned seed = time (NULL);
    cout << "seed = " << seed << endl;
    srand (seed);
    int min, max;
    cout << "Enter minimum and maximum: ";
    cin >> min >> max;      // lowest and highest numbers
    int range = max - min + 1 ;    // number of numbers in range
    for (int i = 0; i < 20; i++) {
        int r = rand () /100%range + min;
        cout << r << " ";
    }
    cout << endl;
}
```

```
seed = 808237677
Enter minimum and maximum : 1 100
85 57 1 10 5 73 81 43 46 42 17 44 48 9 3 74 41 4 30 68
```

```
seed = 808238101
Enter minimum and maximum : 22 66
63 29 56 22 53 57 39 56 43 36 62 30 41 57 26 61 59 26 28
```

في أول تنفيذ للبرنامج تم توليد 20 عدداً صحيحاً موزعة بإنتظام في الفترة بين 1 و 100 . في التنفيذ الثاني تم توليد 20 عدداً صحيحاً موزعة بإنتظام في الفترة بين 22 و 66 .

في الحلقة التكرارية for قسمت الدالة () rand على 100 لتلغي آخر عددين في أقصى اليمين للرقم العشوائي . لقد تم ذلك لتعويض مشكلة أن مولد الأرقام العشوائية هذا على وجه الخصوص يولد أرقام تتغير بين فردي وزوجي . لذلك فإن التعبير rand () / 100%range ينتج أرقاماً عشوائية في الفترة من 0 إلى range - 1 و التعبير rand () / 100%range + min ينتج أرقاماً عشوائية في الفترة من min إلى max .

أفضل طريقة لتوليد الأرقام العشوائية موضحة في المسألة 20.8 .

أسئلة للمراجعة

1.3 ما هو أقل عدد من التكرار

أ – يمكن أن يتم بالحلقة التكرارية while ؟

ب – يمكن أن يتم بالحلقة التكرارية while ... do ؟

2.3 ما الخطأ في الحلقة التكرارية الآتية :

```
while (n <= 100)
sum += n*n;
```

3.3 إذا كانت s جملة مركبة و e1 و e2 و e3 تعبيرات فما الفرق بين جزئية البرنامج التالية :

```
for (e1; e2; e3)
s;
```

وجزئية البرنامج التالية

```
e1;
while (e2) {
    s;
    s3;
}
```

4.3 ما الخطأ في البرنامج التالي :

```
main ()
{
    const double PI;
    int n ;
    PI = 3.14159265358979 ;
    n = 22;
}
```

5.3 ما المقصود بـ تكرار لا نهائي "infinite loop" وكيفية الاستفادة منه ؟

6.3 كيف يمكن بناء حلقة تكرارية بحيث تنتهي بجملة داخل البلوك الخاص بها ؟

7.3 لماذا يجب تجنب إختبارات التساوي مع المتغيرات ذات النقطة العشرية المتنقلة؟

مسـائل محلولـة

8.3 تتبع الجمل الآتية مبيناً قيمة كل متغير في كل مرة

```
int x, y, z;
x = y = z = 6 ;
x *= y += z -= 4;
```

في البداية تم تخصيص العدد 6 لكل من المتغيرات z و y و x . بعد ذلك نقص المتغير z بمقدار 4 فأصبحت قيمته 2 . بعد ذلك زادت قيمة y بمقدار 2 فأصبحت قيمته 8 . عند ذك ضرب x في 8 فأصبحت قيمتها 48.

9.3 نفترض أن e هي تعبير و s هي جملة والمطلوب تحويل كل من الحلقات التكرارية for الآتية إلى ما يناظرها من الحلقات التكرارية while .

a. for (; e ;) s

b. for (; ; e) s

a. while (e) s ;

b. while (1) { s ; e ; } ,

بفرض أن s لا تحتوي على عبارة continue (إنظر المسألة 3.3) .

10.3 حول الحلقة التكرارية for إلي حلقة تكرارية while :

```
for (int i = 1; i <= n ; i++)
    cout << i*i;

int i = 1;
while (i <= n)
    cout << i*i;
    i++;
}
```

11.3 صف شكل الخرج من البرنامج التالي :

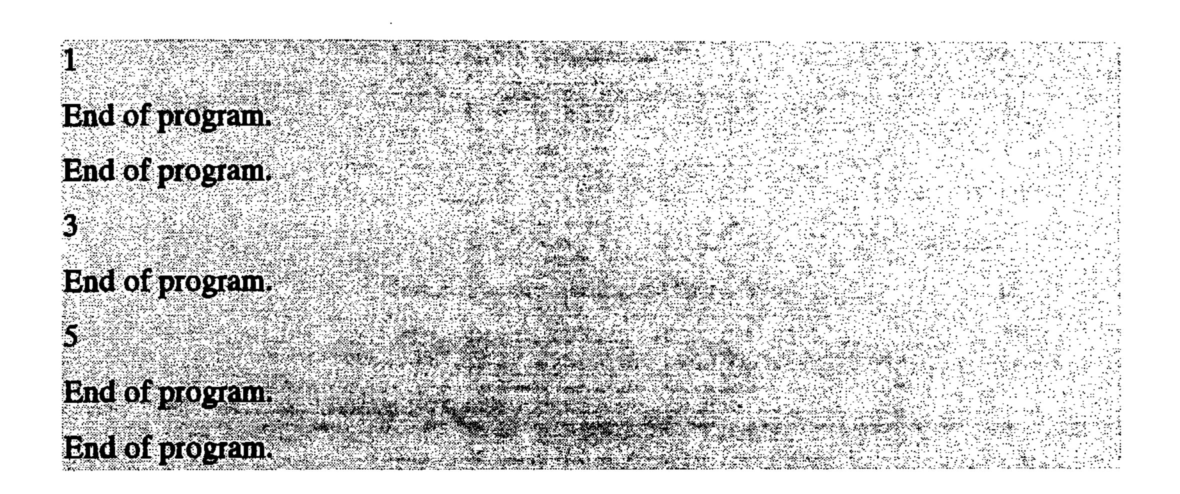
```
main ()
{
    for (int i = 0 ; i< 8; i++)
        if (i%2 == 0) cout << i + 1 << endl;
        else if (i%3 == 0) cout << i*i << endl;
        else if (i%5 == 0) cout << 2*i - 1 << endl;
        else cout << i << endl;
}
```

1 1 3 9 5 9 7 7

12.3 صف شكل الخرج من البرنامج التالي :

```
main ()
{
    for (int i = 0; i < 8 ; i++) {
        if (i%2 == 0) cout << i + 1 << endl;
        else if (i%3 == 0) continue;
```

```
        else if (i%5 == 0) break;
        cout << "End of program. \n" ;
    }
    cout << "End of program. \n" ;
}
```



```
1
End of program.
End of program.
3
End of program.
5
End of program.
End of program.
```

13.3 في النوع float الذي يحتوي على 32 بت و 23 بت يتم استخدامها لتخزين الجزء العشري و 8 بت تستخدم لتخزين الأس .

أ – كم عدد الخانات الناتجة عن النوع float الذي يحتوي على 32 بت ؟

ب – ما مدى المقدار الذي يحتوي على 32 بت من النوع float ؟

أ – الـ 23 بت تبدأ من البت رقم 2 إلى البت رقم 24 في الجزء العشري . البت الأولى يجب أن تكون 1 لذلك فإنها غير مخزنة . وهكذا فإن 24 بت تم تمثيلهم . هذه الـ 24 بت يمكن أن تحفظ 2^{24} من الأرقام. الـ $2^{24} = 16,777,216$ تمثل بـ 7 خانات في المدى الكامل لذلك فإن الـ 7 خانات يمكن تمثيلهم . لكن آخر بت يكون هناك شك فيها بسبب التقريب. وعلى ذلك فالنوع float ذو 32 بت ينتج 6 خانات للدقة.

ب – الـ 8 بت المستخدمة في الأس يمكن أن تحتفظ بـ $2^{8} = 256$ عدد مختلف . إثنين منهم محجوزين لبيان تجاوز الحد الأقصى والحد الأدنى المسموح بهما يتبقى 254 عدد للأس . لذلك فإن مدى الأس يكون من -126 إلى +127 ينتج مقدار في المدى من $2^{-126} = 1.175494 \times 10^{-38}$ إلى $2^{127} = 1.70141 \times 10^{38}$.

مسائل محلولة في البرمجة

14.3 أكتب برنامج يحول البوصات inchs إلى سنتيمترات centimeters . على سبيل المثال لو أن المستخدم أدخل 16.9 بوصة تعبر عن الطول فإن الخرج يكون 42.926 سم . (البوصة الواحدة = 2.54 سم).

سوف نستخدم متغيرين من النوع float :

```
main ()
{
      float inches, cm;
      cout << "Enter length in inches: " ;
      cin >> inches;
      cm = 2.54*inches;
      cout << inches << " inches = " << cm << " centimeters. \n";
}
```

```
Enter length in inches: 16.9
16.9 inches = 42.926 centimeters.
```

البرنامج ببساطة يقرأ الدخل للمتغير inches ويحوله إلى سنتيمترات ويطبع الناتج في المتغير cm ويخرجها.

15.3 أكتب برنامج لإيجاد الجذر التربيعي الصحيح لأي عدد معطى . الجذر التربيعي هو العدد الصحيح الذي يكون تربيعه أقل من أو يساوى العدد المعطى.

نحن نستخدم طريقة حساب شاملة: أوجد كل الأعداد الصحيحة الموجبة التي مربع أي منها أقل من أو يساوي العدد المعطى عند ذلك يكون أكبر هذه الأعداد هو الجذر التربيعي الصحيح :

```
main ()
{
      float x ;
      cout << "Enter a positive number : ";
      cin >> x;
      for (int n = 1; n*n <= x; n++)
            ;                                         // the null statement
      cout << "The integer square root of " << x << " is "
            << n-1 << endl;
}
```

```
Enter a positive number : 1234.56
The integer square root of 1234.56 is 35
```

نبدأ بـ n = 1 ونستمر في زيادة n حتى يكون مربع n أكبر من x ، n*n > x . عندما تنتهي الحلقة التكرارية for تكون n هي أصغر عدد صحيح مربعه أكبر من x لذلك n-1 يكون هو الجذر التربيعي الصحيح لـ x .

لاحظ استخدام الجملة الصفرية (;) بداخل الحلقة for . كل شئ كنا في حاجة إلى عمله في الحلقة تم عمله بداخل أجزاء تحكم الحلقة . لكن الفاصلة المنقوطة مازالت ضرورية في نهاية الحلقة .

16.3 أكتب ونفذ برنامج يستخدم مباشرة عامل القسمة " / " وعامل الباقي % لقسمة أرقام صحيحة موجبة .

الخواريزم المستخدم هنا يطبق على الكسر n/d يكرر طرح d من n إلى أن تصبح n أقل من d . عند هذه النقطة تكون n هي الباقي وعدد مرات التكرار q اللازمة للوصول لها يكون خارج القسمة:

```
main ( )
{
     int  n,  d,  q,  r;
     cout  <<  "Enter numerator : " ;
     cin  >>  n ;
     cout  <<  " Enter denominator : " ;
     cin  >>  d ;
     for  (q  =  0 ,   r  =  n ;    r  >  d ;   q++)   r  -=  d ;
     cout  <<  n  <<  " / "  <<  d  <<  " = "  <<  q  <<  endl ;
     cout  <<  n  <<  " % "  <<  d  <<  " = "  <<  r  <<  endl ;
     cout  <<  " ( "  <<  q  <<  ")  ("  <<  d  <<  ")  +  ("  <<  r  <<  ")  =  "
          <<  n  <<  endl ;
}
```

```
Enter numerator : 30
Enter denominator : 7
30 / 7 = 4
30 % 7 = 2
(4) (7) + (2) = 30
```

تكرار هذا التنفيذ أربع مرات : 30-7=23 ، 23-7=16 ، 16-7=9 ، 9-7=2 . لهذا فإن خارج القسمة هو 4 والباقي 2 . لاحظ أن العلاقة التالية صحيحة دائماً للقسمة على رقم صحيح :

(ناتج القسمة) × (المقام) + (الباقي) = البسط

17.3 اكتب ونفذ برنامج لعكس ترتيب الأرقام في عدد صحيح موجب

الحيلة هنا هي نزع الأرقام واحد بواحد من الرقم المعطي ثم تجميعها في رقم صحيح آخر بترتيب عكسي

```
main ()
{
     long  m,  d,  n  =  0;
     cout  <<  "Enter a positive integer : " ;
     cin  >>  m ;
     while  (m  >  0)  {
          d = m % 10 ;              // d will be the right-most digit of m
          m  /=  10 ;               // then remove that digit from m
          n  =  10*n + d ;          // and append that digit to n
     }
     cout  <<  "The reverse is "  <<  n  <<  endl ;
}
```

```
Enter a positive integer : 123456
The reverse is 654321
```

في هذا التنفيذ ، تبدأ m بالقيمة 123456 . في حلقة التكرار الأولي تعطي d الرقم 6 وتقل قيمة m إلى 12,345 وتزاد إلى 6 . وفي الدورة الثانية ، تعطي d الرقم 5 بينما تقل قيمة m إلى 1234 وتزاد قيمة n إلى 65 . في الدورة الثالثة تعطي d الرقم 4 وتقلل m إلى 123 وتزاد قيمة n إلى 654 . ويستمر ذلك حتى الدورة السادسة حيث تعطي d الرقم 1 . وتقلل m إلى صفر وتزاد n إلى 654321.

18.3 أعد كتابة الحلقة التكرارية for من المثال 6.3 مستخدماً عامل التعبير الشرطي مكان الأمر if.

التعبير الشرطي (n < min ? n : min) يصبح n إذا كانت n < min ، ويصبح min عدا ذلك. لذلك فإن اعطاء هذه القيمة إلى min يكافئ السطر الأول في الحلقة التكرارية for بالمثال . بالمثل اعطاء القيمة max = (n > max ? n : min) يماثل السطر الثاني من حلقة for التكرارية الأخرى .

```
for (min  =  max  =  n;  n  >  0 ;)    {
     min   =  (n  <  min  ?  n  :  min) ;          //  min and max are the smallest
     max  =  (n  >  max  ?  n  :  min) ;          //  and largest of the n that
     cin  >>  n ;                                 //  have been read so far
}
```

لاحظ في هذا النموذج أننا لم نستخدم ما يكافئ else if

19.3 طبق الخواريزم الاقليدي لايجاد القاسم المشترك الأعظم لرقمين صحيحين موجبين .

يحول الخواريزم الاقليدي رقمين صحيحين موجبين (m , n) إلى رقمين (d , 0) بالقسمة المتتالية للرقم الأكبر على الرقم الصحيح الأصغر مع استبدال الأكبر بالباقي. عندما يصبح الباقي صفراً فإن الرقم الآخر يصبح هو القاسم المشترك الأعظم للرقمين الأصليين (وكذلك للأزواج المتوسطة من الأرقام).

كمثال إذا كانت m = 532 ، n = 112 فإن الخواريزم الاقليدي يخفض الرقمين (532 ، 112) إلى (صفر، 28) كالتالي

(532 ، 112) ⟶ (112 ، 84) ⟶ (84 ، 28) ⟶ (28 ، 5)

ولذلك فإن 28 هو القاسم المشترك الأعظم بين 532 ، 112 يمكن التحقق من هذه النتيجة حيث أن 532=19x28 ، 112=28x4 إن جدوى الخواريزم الاقليدي هنا هو أن كل زوج في التتابع له نفس مجموعة القواسم التي هي بالضبط مجموعة المعاملات للقاسم المشترك الأعظم. في المثال السابق تلك المجموعة هي {1 ، 2 ، 4 ، 7 ، 14 ، 28} . ان السبب في عدم تغير هذه المجموعة من القواسم خلال عملية الاختزال هو أنه عندما m=n.q+r فإن الرقم يكون قاسم مشترك لكل من m ، n إذا كان فقط هو قاسم مشترك بين n ، r.

```
main ()
{                                              // begin scope of main ()
    int m, n, r;
    cout << "Enter two positive integers : " ;
    cin >> m >> n ;
    if (m < n)  { int temp = m;  m = n ; n = temp ; }   // make m >= n
    cout << "The g.c.d. of " << m << " and " << n << " is " ;
    while (n > 0)
        r = m % n ;
        m = n;
        n = r;
    }
    cout << m << endl ;
}
```

```
Enter two positive integers : 532   112
The g.c.d. of 532 and 112 is 28
```

20.3 أكتب واختبر برنامجاً يقرأ عدد من أزواج الأرقام الحقيقية (x ، y) ثم يحسب المعادلة التقريبية بأقل المربعات على هيئة خط لمجموعة من النقاط. باستخدام المعادلة y=mx+b حيث

$$m = \frac{(\sum xy) - \bar{y}(\sum x)}{(\sum xx) - \bar{x}(\sum x)}$$

$$b = \bar{y} - m\bar{x}$$

حيث $\bar{x}$ هو متوسط قيم x للنقط المعطاة.

نستعمل التدقيق المزدوج double precision للأرقام الحقيقية لتقليل الخطأ الناتج عن التقريب.

```
main ()
{
    int n ;                   // number of data points
    double x, y, sumX = 0.0, sumY = 0.0 , sumXX = 0.0, sumXY=0.0;
    cout << "How many points : " ;
    cin >> n ;
    cout << "Enter " << n << " pairs, one pair per line:\n" ;
    for (int i = 1; i <= n ; i++) {
        cout << '\t' << i << " : " ;
        cin >> x >> y ;
        sumX += x; // accumulate the sum of the x' s in sumX
        sumY += y; // accumulate the sum of the y' s in sumY
        sumXX += x*x; // accumulate the sum of the x*x in sumXX
        sumXY += x*x; // accumulate the sum of the x*y in sumXY
    }
    double meanX = sumX/n ;
    double meanY = sumY/n ;
    double m = (sumXY - meanY*sumX) / (sumXX - meanX*sumX) ;
    double b = meanY - meanX*m ;
    cout << "The equation of the regression line is :\n"
         << "\ty = " << m << "x + " << b << endl ;
}
```

```
How many points : 4
Enter 4 pairs, one pair per line :
    1 : 1.0 5555.04
    2 : 2.0 6666.07
    3 : 3.0 7777.05
    4 : 4.0 8888.09
The equation of the regression line is :
y = 1111.01x + 4444.03
```

كل من عمليات الجمع الأربعة sumX ، sumY ، sumXX ، sumXY تتراكم خلال حلقة الدخل. بعدها تحسب المتوسطات meanX ، meanY ثم تستخدم في المعادلة لحساب الميل m والتقاطع مع المحور y ، b للخط التقريبي.

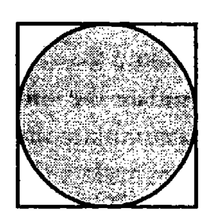
ان خرج هذا البرنامج مفيد جداً. ان الخط التقريبي يتفق إلى اقرب درجة مع البيانات المعطاة. ذلك الخط في حالة البيانات المعطاة يكون على الصورة

$$y = 1111.01x + 4444.03$$

وهو أقرب خط ممكن بمعنى أن مجموع مربعات الفرق في اتجاه y والنقط المعطاة يكون حد أدنى. قيمة هذه النتيجة هو امكانية استخدامها للتطابق والامتداد. كمثال لتخمين قيمة y التي تناظر قيمة x=3.2 نعوض ببساطة هذه القيمة في المعادلة

$$y = 1111.01(3.2) + 4444.03 = 7999.26$$

21.3 استخدام طريقة محاكاة مونت كارلو لحساب النسبة التقريبية π . لقد سميت طريقة محاكاة مونت كارلو بعد الملهي في موناكو. انها تتكون من سحب نقاط عشوائياً وعد تلك النقاط التي تفي بشرط معين. يمكن استخدامها لحساب قيمة π بمحاكاة القاء اسهم على لوحة دائرية مثبتة على مربع.



إذا كانت الأسهم متساوية الاحتمال في الاصطدام بأي نقطة في المربع فإن النسبة التي تصطدم داخل الدائرة تقترب من النسبة بين مساحة الدائرة إلى مساحة المربع . إذا كان للمربع جوانب طولها 2.0 فإن تلك النسبة تكون

$$(\pi r^2)/(s^2) = \pi/4$$

أي أن أربعة أمثال تلك النسبة يقترب من π

من الأسهل استخدام ربع دائرة نصف قطرها الوحدة وتقع في الربع الأول كما هو موضح



بهذه الطريقة فإن الاحداثيات المختارة عشوائياً ستكون جميعها خلال النطاق 5.5 إلى 1.5 وتكون مساحة المربع هي الوحدة ومساحة ربع الدائرة هي $\pi/4$ أي أن النسبة لا تزال $\pi/4$

```
#include <iostream.h>
#include <stdlib.h>
#include <time.h>

main ()
{
    const long int TOSSES = 1000;          // toss 1000 darts
    long int hits = 0;
    float x , y ;
    unsigned seed = time (NULL) ;
    srand (seed) ;
    for (long int i = 0 ; i < TOSSES ; i++) {
        x = float (rand ()) / RAND_MAX;
        y = float (rand ()) / RAND_MAX;
        if (x*x + y*y < 1) ++hits;
    }
    cout << 4.0*hits/TOSSES << endl ;
}
```

```
3.142
```

```
3.13504
```

كلا التنفيذين يعطي تقدير للنسبة π وهو مضبوط لأقرب ثلاث أرقام عشرية. يمكن تحسين الدقة بإلقاء مزيد من الأسهم ولكن على حساب الزيادة في وقت التنفيذ للبرنامج.

22.3 قلد لعبة Monty Hall

سميت لعبة Monty Hall بعد مقدم العاب التليفزيون حيث يمكن للمتنافس أن يفوز بسيارة جديدة بتخمين الباب الذي تقف السيارة وراءه. أصبحت تلك اللعبة محبوبة في التسعينيات لأن أحسن مبدأ للعب هو عكس التوقعات . يختار المتنافس باب واحد وعندها يفتح Monty أحد الأبواب الأخرى التي ليس خلفها السيارة وعند ذلك فإن المتنافس له خيار تغيير اختياره لباب ثالث. يندهش معظم الناس ليتعلموا أن المتنافس له حظ مضاعف ليكسب السيارة إذا قام بتغيير اختياره. يمكن توضيح ذلك بواسطة الاحتمالات المشروطة ولكن لغالبية الناس يكون التمثيل بالحاسب أكثر إقناعاً .

```
#include <iostream.h>
#include <stdlib.h>
#include <time.h>

main ()
{
    cout << "This is the Monty Hall Game. \nYou see three doors "
         << "befor you. One of them has a new car behind it .\n"
         << "you will choose one of the doors. Then, befor you "
         << "get to see which\ndoor has the car behind it , Monty "
         << "will give you the chance to change\nyour choice after "
         << "showing you that one of the other doors has\nnothing "
         << "behind it . \n " ;
    unsigned seed = time (NULL) ;
    srand (seed) ;
    int car, choice, open, option;
    car = rand ()%3 + 1;                         // random integer from 1 to 3
    cout << "Which door do you choose (1 | 2 | 3) :  " ;
    cin >> choice ;
    if (car == 1 && choice == 1)   { open = 3; option = 2; }
    if (car == 1 && choice == 2)   { open = 3; option = 1; }
    if (car == 1 && choice == 3)   { open = 2; option = 1; }
    if (car == 2 && choice == 1)   { open = 3; option = 2; }
    if (car == 2 && choice == 2)   { open = 1; option = 3; }
    if (car == 2 && choice == 3)   { open = 1; option = 2; }
    if (car == 3 && choice == 1)   { open = 2; option = 3; }
    if (car == 3 && choice == 2)   { open = 1; option = 3; }
    if (car == 3 && choice == 3)   { open = 2; option = 1; }
    cout << "Monty shows that there is no car behind door number "
         << open << " . \nDo you want to change your choice to door "
         << "number " << option << " ? (y|n) : ;
    char answer ;
    cin >> answer ;
    if (answer == 'y' || answer == 'y') choice = option ;
    cout << "Door number " << car << " has the car behind it . \n "
    << "Since your final choice was door number " << choice ;
    if (choice == car) cout << " , you won the car !\n " ;
    else cout << " , you did not win. \n " ;
}
```

هذه هي لعبة Monty Hall .

أنك ترى أمامك ثلاثة أبواب . أحداهم خلفه سيارة جديدة . انك ستختار أحد الأبواب وقبل أن تعلم أي الأبواب خلفه السيارة سيعطيك Monty الفرصة لتغير اختيارك بعد أن يعلمك أن أحد الأبواب الأخرى ليس خلفه السيارة أي باب ستختار (1 | 2 | 3) : 3

سيظهر Monty أنه لا توجد سيارة خلف الباب رقم 1 . هل تريد تغيير اختيارك للباب رقم 2 ؟ (y/n): n

الباب رقم 2 خلفه السيارة

حيث أن اختيارك النهائي هو الباب رقم 3 فانك لم تكسب .

23.3 طبق خواريزم البابليون لحساب الجذر التربيعي للرقم 2 .

خواريزم البابليون (سمي هكذا لأنه كان يستخدم بواسطة البابليون القدامي) لحساب $\sqrt{x}$ بالتبديل المتكرر لتقدير x بالتقدير الأقرب (x+2/x)/2 . لاحظ أن هذه ببساطة المتوسط لـ x ، 2/x .

```
#include <iostream.h>
#include <math.h>    // needed for the fabs () function

main ()
{
    const double TOLERANCE = 5e - 8 ;
    double x = 2.0 ;
    while (fabs (x*x - 2.0) > TOLERANCE) {
        cout << x << endl ;
        x = (x + 2.0/x)/2.0 ;      // average of x and 2/x
    }
    cout << "x = " << x << ", x*x = " << x*x << endl ;
}
```

```
2
1.5
1.41667
1.41422
x = 1.41421,   x*x
```

نستخدم هنا السماح بقيمة $5e^{-8}$ = 0.00000005 لضمان دقة حتى 7 خانات عشرية. الدالة fabs () (عن القيمة المطلقة للرقم الحقيقي) والمعرفة في الملف التقديمي <math.h> تعيد القيمة المطلقة للتعبير المرسل لها. ويستمر التكرار حتى تصبح x*x في حدود المسموح به من 2 .

مسائل إضافية

24.3 حول حلقة for التالية إلى حلقة while

```
for (int i = 20 ; i > 10 ; i --)
cout << i*i ;
```

25.3 نفذ البرنامج في المثال 13.3 لإيجاد أطوال أنواع ++C الاساسية الاثنى عشر في نظامك.

26.3 نفذ البرنامج في المثال 14.3 لإيجاد السماح ومدى المقدار للأرقام الحقيقية في نظامك.

27.3 إوصف الخرج من الجزء التالي

```
int f0 = f1 = f2 = 1 ;
for (int i = 0; i < 10 ; i ++) {
    f0 = f1 ;
    f1 = f2 ;
    f2 = f0 + f1 ;
    cout << f2 < endl ;
}
```

28.3 صف الخرج الناتج عن الجزء التالي :

```
for (int i = 0 ; i < 8 ; i ++)
    if (i%2 == 0) cout << i + 3 << endl ;
    else if (i%3 == 0) cout << 2*i - 1 << endl ;
    else if (i%5 == 0) cout << i*i << endl ;
    else cout << i << endl ;
```

29.3 صف الخرج الناتج عن الجزء التالي :

```
int i = 0 ;
while (++i <= 9) {
    if (i == 5) continue ;
    cout << i << endl ;
}
```

30.3 صف الخرج الناتج عن الجزء التالي :

```
int i = 0 ;
while (i < 5) {
    if (i < 2) {
        i += 2;
        continue ;
    }
    else cout << ++i << endl ;
    cout << "Bottom of loop .\n" ;
}
```

31.3 في النوع double 64 بت . يستخدم 52 بت لتخزين الرقم بينما تستخدم 11 بت لتخزين الأس.

أ – كم دقة الأرقام التي تمثلها 64 بت من النوع double ؟

ب – ما هو مدى المقدار الذي يمثله 64 بت من النوع double؟

مسائل اضافية للبرمجة

32.3 أكتب برنامجاً ليقرأ درجة الحرارة بالتقدير المئوي ويطبع المكافئ بدرجات الفهرنهايت. كمثال إذا ادخل المستخدم 75.4 درجة مئوية يكون الخرج 135.72 بالتقدير الفهرنهايتي.

33.3 أكتب برنامج يحول السنتيمتر إلى البوصة . كمثال إذا ادخل المستخدم 52.7 لطول بالسنتيمتر يكون الخرج 20.748 بوصة.

34.3 اكتب برنامج يحول الأوزان من الرطل إلى الكيلوجرام . كمثال إذا ادخل المستخدم 160 للوزن بالرطل يكون الخرج 72.5748 كجم (الرطل يساوي 0.453592 كيلو)

35.3 اكتب برنامج ليقرأ نصف قطر كرة ويطبع مساحة سطحها وحجمها.

36.3 عدل ونفذ البرنامج في المثال 1.3 حتى يطبع أيضاً مربع لـ n-1 ومربع n .

37.3 عدل البرنامج في المثال 1.3 حتى يستخدم الحلقة do ... while .

38.3 عدل البرنامج في المثال 1.3 حتى يستخدم الحلقة for .

39.3 اكتب ونفذ برنامجاً مثل الذي بالمثال 2.3 والذي يطبع المجموع أول n تكعيب .

40.3 عدل البرنامج بالمثال 3.3 لتستخدم الحلقة while لحساب المضروب .

41.3 عدل البرنامج بالمثال 3.3 لتستخدم الحلقة for لحساب المضروب.

42.3 عدل البرنامج بالمثال 3.3 لتستخدم الحلقة do ... while لحساب المضروب مع استخدام متغير تحكم بداخل الحلقة والتي تزيد n بدلاً من انقاصها.

43.3 عدل البرنامج بالمثال 3.3 لتستخدم الحلقة do ... while لحساب المضروب مستخدماً متغير للتحكم داخل الحلقة والذي يزيد n بدلاً من انقاصه.

44.3 عدل البرنامج بالمثال 3.3 لتستخدم الحلقة for مستخدماً متغير للتحكم داخل الحلقة ليزيد n بدلاً من انقاصها .

45.3 اكتب ونفذ برنامجاً ليقرأ عدد صحيح موجب n ويقرأ عدد n أرقام صحيحة اضافية ويطبع مجموعها استخدم حلقة do ... while .

46.3 اكتب ونفذ برنامجاً ليقرأ عدد صحيح موجب n ثم يقرأ عدد n من الأرقام الصحيحة الأخرى ثم يطبع مجموعها . استخدم حلقة التكرار for

47.3 اكتب ونفذ برنامجاً ليقرأ تتابعاً من الأرقام الصحيحة إلى أن يدخل رقم سالب عندها يطبع مجموع الأرقام الموجبة.

48.3 عدل البرنامج في المثال 14.3 ليطبع الدقة ومدى المقدار للنوع long double ببساطة بدل float بـ long double ، و FLT بـ LDBL .

49.3 اكتب ونفذ برنامجاً ليقرأ رقم صحيح موجب عندها يطبع مثلث من النجوم في نفس هذه السطور . استخدم الحلقة for . كمثال اذا كانت n = 4 فإن الخرج يكون على الشكل :

```
*
**
***
****
```

50.3 اكتب ونفذ برنامجاً ليقرأ عدد صحيح موجب n ثم يطبع معين من النجوم على 2n-1 من الصفوف استخدم الحلقة for . كمثال إذا كانت n=4 فإن الخرج يكون على الشكل

```
   *
  ***
 *****
*******
 *****
  ***
   *
```

51.3 اكتب ونفذ برنامجاً يستخدم مباشرة عامل القسمة / وعامل الباقي % لقسمة رقم صحيح سالب على رقم صحيح موجب . أنظر المسألة 16.3 والمثال 16.1 .

52.3 أعد حل المسألة 19.3 مستخدماً الحلقة do ... while بدلاً من الحلقة while .

53.3 عدل برنامج الجذر التربيعي للرقم الصحيح بالمسألة 15.3 حتى تنفذ بأكثر كفاءة . استخدم خوارزم البحث الثنائي بدلاً من خوارزم البحث الخطي . أولاً اختبر ما إذا كان الرقم الصحيح الموجب المعطى x أقل من 9 ، إذا لم يكن اطبع اما 0 (إذا كانت x<1) أو 1 (إذا كانت x<4) أو 2 ، ثم نعود . إذا كانت x≥9 فإن الجذر التربيعي الصحيح لها يكون بين 2 ، x/2 . اقسم هذه الفترة ثم قارن x بـ n*n حيث n هي منتصف تلك الفترة . استخدم المقارنة لتحديد مكان المنتصف الذي يحتوي على الحل. كرر هذه العملية على منتصف المسافات . استخدم الارقام الصحيحة فقط للنقط الطرفية والمنتصف للمسافات. عندما تكون نقطة المنتصف أكبر من النهاية اليسرى بما يساوى 1 فإنها تكون الحل.

55.3 عدل البرنامج بالمثال 14.3 ليطبع الدقة ومدى المقدار للنوع double

56.3 عدل برنامج المعادلة التربيعية في المسألة 20.2 ليطبع المعادلة على شكل استخدامها في الحساب . كمثال إذا كانت a=1 ، b=0 و c=-3 فإن البرنامج يطبع x^2-3=0 بدلاً من 1x^2+0x+-3=0 .

57.3 عدل برنامج المعادلة التربيعية في المسألة 20.2 حتى تحل المعادلة في الحالات الخاصة عند a=0 ، b=0 أو c=0 . كمثال إنها تشير إلى أن 1.25 هو الحل للمعادلة 4x-5=0 ، وأن صفر هو حل المعادلة 4x=0 وأنه ليس هناك حل للمعادلة 5=0 وأن كافة الأعداد الحقيقية هي حلول للمعادلة 0=0 .

58.3 اكتب واختبر برنامج ليدخل ثلاث أرقام صحيحة موجبة day و month و year ثم يطبع التاريخ الذي يعبر عنه وعدد الأيام في ذلك الشهر وجملة عما إذا كانت تلك السنة كبيسة . كمثال إذا كانت الأرقام الثلاث هي 6 ، 4 ، 1997 عندئذ يطبع البرنامج April 6, 1997 (بدلاً من 4/6/97) شهر ابريل به 30 يوم وسنة 1997 ليست كبيسة.

59.3 اكتب واختبر برنامجاً ليدخل أربعة أرقام صحيحة وموجبة day ، month ، year ، days ثم يطبع تاريخين . تاريخ معبر عن day ، month ، year والتاريخ الذي يكون بعد days .

كمثال إذا كانت الأرقام الأربعة هي 6 ، 4 ، 1997 ، 100 حينئذ فإن التاريخين المطبوعين يكونان April 6, 1997 (عن 4/6/97) و July 15 , 1997 (عن 4/6/97 +100 days) .

60.3 عدل برنامج التوافق الخطي (المسألة 20.3) بحيث أنه بعد حساب معادلة خط التوافق ، يسمح للمستخدم بحساب احداثيات بينية بإدخال قيم x واخراج قيم y المناظرة والمحسوبة من المعادلة.

61.3 عدل لعبة Monte Hall (مسألة 22.3) بحيث يمكن للمستخدم اللعب بصورة متكررة في تنفيذ واحد للبرنامج . عدْ عدد مرات المكسب للاعب ويطبع النسبة المئوية للفوز في نهاية البرنامج.

62.3 عدل برنامج خوارزم البابليون (المسألة 23.3) لكي يحسب الجذر التربيعي لرقم موجب t يتم ادخاله تفاعلياً . احسب متوسط التدرج x مع x+t/x)/2) .

اجابات لأسئلة المراجعة

1.3 الحد الأدنى للتدرجات عندما

أ - حلقة while تعمل عدد 0 .

ب - حلقة do ... while يمكن أن تعمل عدد 1 .

2.3 انها حلقة لا نهائية . متغير التحكم n لا يتغير .

3.3 ليس هناك فرق بين هاتين المقطعين إلا إذا احتوت S على خبر continue . كمثال حلقة for التالية ستتدرج أربع مرات ثم تنتهي كالمعتاد . ولكن الحلقة while ستكون حلقة لا نهائية .

```
for (i = 0;  i < 4;  i++)
    if (i == 2) continue ;

i = 0 ;
While (i < 4)  {
    if (i == 2) continue ;
    i ++ ;
}
```

4.3 الثابت PI لم يتم اعطاؤه قيمة ابتدائية . كل الثوابت لابد من اعطاءها قيم ابتدائية عند الإعلان عنها .

5.3 "حلقة لا نهائية" هي التي لا تنتهي . عامة تعتبر هذه الحلقة عادة برمجة سيئة لأن البرنامج الذي يحتويها ينتهي بصورة عادية . ومع ذلك فإن حلقة لانهائية ظاهرة مثل التالية يمكن أن تكون مفيدة:

```
while (1) {
    cin >> n ;
    if (n == 0) break ;
    process (n) ;
}
```

إن جملة break ستنهي الحلقة بمجرد ادخال 0 . هذه مفيدة لأنها تسمح للخطوات بأن تكون مختصرة بعض الشيء عما إذا كانت الحالة (n == 0) ستستعمل مع جملة while .

6.3 جملة break يمكن استخدامها لإنهاء حلقة من خلال منتصف منطقتها . المثال السابق يوضح هذه الطريقة.

7.3 نظراً لخطأ التقريب القيمة المضبوطة لرقم float أو double من المستبعد أن تكون كما نتوقع . ولذلك فإن شرطاً مثل

```
if (z == c)  ....
```

يجب تجنب استخدامه.

الفصل الرابع

4 الـــــدوال
Functions

معظم البرامج المفيدة أكبر بكثير من البرامج التي ذكرناها إلى الآن . لعمل برامج كبيرة يسهل تتبعها يقوم المبرمجين بتقسيم البرامج الرئيسية إلى برامج فرعية sub programs . هذه البرامج الفرعية تسمى دوال functions . يمكن ترجمة وإختبار البرامج الفرعية كل على حدة وإعادة إستخدامها في برامج مختلفة .

1.4 دوال مكتبة C القياسية

مكتبة لغة C القياسية هي مجموعة من الدوال المعرفة مسبقاً وبعض عناصر البرامج التي يمكن الوصول إليها من خلال ملفات الرأس . لقد استخدمنا سابقاً بعض من هذه الملفات . الثابت INT_MAX معرف في الملف <limits.h> (مثال 14.1) والدالة () rand معرفة في الملف <stdlib.h> (مثال 21.3) والدالة () time معرفة في الملف <time.h> (مثال 22.3) . الدوال الرياضية الشائعة معرفة في الملف <math.h> . مثالنا الأول يوضح استخدام واحدة من هذه الدوال الرياضية.

مثال 1.4 دالة الجذر التربيعي () sqrt

الجذر التربيعي لعدد موجب معلوم هو العدد الذي مربعه ذلك العدد المعلوم .

الجذر التربيعي للعدد 9 هو 3 لأن مربع 3 هو 9 . نحن نستطيع أن نعتبر أن دالة الجذر التربيعي كصندوق مغلق . عندما تضع بداخله 9 يخرج منه 3 . عندما يكون الرقم 2 دخل يكون الخرج هو 1.4121 . هذه الدالة لها نفس طبيعة عمل البرنامج الكامل ، دخل ـ إجراء عملية على الدخل ـ خرج . على أي حال خطوة إجراء العملية في الدالة تكون مستترة . ونحن لسنا في حاجة لمعرفة ما تفعله الدالة للعدد 2 لتنتج العدد 1.41421 . كل ما نحتاج لمعرفته هو أن الخرج 1.41421 له خاصية الجذر التربيعي : مربع الخرج هو الدخل 2.

هذا برنامج مبسط يستخدم دالة الجذر التربيعي المعرفة مسبقاً :

```
#include <iostream.h>
#include <math.h>
//   Test - driver for the sqrt function:
main ()
{
     for (int i = 0; i < 6 ; i++)
          cout << "\t" << i << "\t" << sqrt (i) << endl;
}
```

```
0    0
1    1
2    1.41421
3    1.73205
4    2
5    2.23607
```

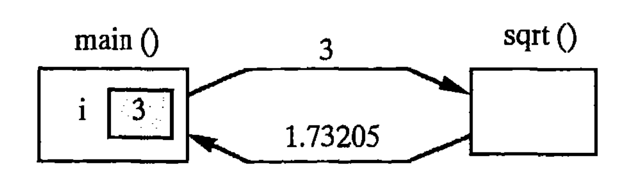
هذا البرنامج يطبع الجذور التربيعية للأعداد من 0 إلى 5 . كل مرة يقدر التعبير sqrt () في الحلقة التكرارية for وتنفذ الدالة sqrt . برنامج الدالة الحقيقي مختفي بعيداً في داخل مكتبة لغة سي القياسية . باستخدام هذه الدالة سوف يستبدل التعبير sqrt (i) بالجذر التربيعي الحقيقي لقيمة i في نفس اللحظة .

لاحظ التوجيه <math.h> # include على السطر الثاني . هذا ضروري للمترجم ليجد تعريف للدالة sqrt () . فهو يخبر المترجم بأن الدالة قد أعلن عنها في ملف الرأس <math.h>.

دالة مثل sqrt () نفذت باستخدم إسمها كمتغير في جملة كالتالي :

```
y = sqrt (x) ;
```

هذا يسمى تنفيذ "invoking" أو استدعاء "calling" الدالة . لذلك في المثال 1.4 فإن الأمر sqrt (i) يستدعي الدالة sqrt . التعبير i الذي بين الأقواس يسمى معامل الدالة (argument or parameter) ونقول أن المتغير أرسل بقيمة إلى الدالة . لذلك عندما تكون i تساوي 3 ترسل إلى الدالة sqrt بالاستدعاء sqrt (i) . هذه العملية موضحة بالشكل التالي :



المتغير i أعلن عنه في البرنامج الرئيسي () main . أثناء التكرار الرابع للحلقة for كانت قيمة i تساوي 3. هذه القيمة أرسلت إلى الدالة () sqrt التي أرجعت القيمة 1.73205 إلى البرنامج الرئيسي.

مثال 2.4 إختبار تطابق دوال المثلثات

هذا برنامج آخر يستخدم ملف الرأس <math.h> . الغرض منه التحقق من تطابق الدالة المثلثية القياسية $\sin 2x = 2 \sin x \cos x$:

```
#include <iostream.h>
#include <math.h>
//   program to test trigonometric identity  sin 2x = 2 sin x  cos x:
main ()
{
    for (float x = 0; x < 2; x += 0.2)
        cout << "\t" << x << "\t\t" << sin (2*x) << "\t"
             << 2*sin (x) * cos (x) << endl;
}
```

0	0	0
0.2	0.389418	0.389418
0.4	0.717356	0.717356
0.6	0.932039	0.932039
0.8	0.999574	0.999574
1	0.909297	0.909297
1.2	0.675463	0.675463
1.4	0.334988	0.334988
1.6	-0.0583744	-0.0583744
1.8	-0.442521	-0.442521

البرنامج يطبع x في العمود الأول و sin 2x في العمود الثاني و 2 sin x cos x في العمود الثالث . لكل قيمة لـ x يتم إختبار sin 2x = 2 sin cos x . بالطبع هذا ليس إثبات للتطابق . ولكنه يعطي إقناع واضح لحقيقة التطابق.

قيم الدالة يمكن أن تستخدم في التعبير الجبري مثل المتغيرات العادية . لذلك يمكن أن نكتب التالي :

```
y = sqrt (2);

cout << 2*sin (x) * cos (x);
```

يمكن أيضاً تفريع إستدعاء الدالة كالتالي :

```
y = sqrt (1 + 2*sqrt (3 + 4*sqrt (5)))
```

معظم الدوال الحسابية الموجودة على الآلة الحاسبة معلن عنها في ملف الرأس <math.h> بما فيها الدوال المبينة في الجدول 1.4 .

جدول 1.4 بعض دوال الملف math.h

الدالة	الوصف	مثال
acos (x)	معكوس (مقلوب) الدالة cosin x (تقدير دائري)	acos (0.2) ترجع 1.36944
asin (x)	معكوس الدالة sine x (تقدير دائري)	asin (0.2) ترجع 0.201358
atan (x)	معكوس الدالة tan x (تقدير دائري)	atan (0.2) ترجع 0.197396
ceil (x)	سقف الرقم x	ceil (3.141593) ترجع 4.0
cos (x)	cosine x (بالردين)	cos (2) ترجع -0.416147
exp (x)	حساب الدالة الأسية لـ x (للأساس e)	exp (2) ترجع 7.38906
fabs (x)	القيمة المطلقة لـ x	fabs (-2) ترجع 2.0
floor (x)	أرضية الرقم x	floor (3.141593) ترجع 3.0
log (x)	لوغاريتم x (الأساس e)	log (2) ترجع 0.693147
log10 (x)	اللوغاريتم للعدد x (الأساس 10)	log10 (2) ترجع 0.30103
pow (x, p)	حساب x^p	pow (2, 3) ترجع 8.0
sin (x)	sine x (بالردين)	sin (2) ترجع 0.909297
sqrt (x)	الجذر التربيعي للعدد x	sqrt (2) ترجع 1.41421
tan (x)	الدالة tan x (بالتقدير الدائري)	tan (2) ترجع -2.18504

لاحـظ أن كل دالة حسـابيـة ترجع رقم من النوع double . لـو أن العـدد الداخل للدالة من النوع الصحيح int فإنه يرقى إلى النوع double قبل حساب خرج الدالة.

الجدول 2.4 يبين بعض ملفات الرأس الهامة في مكتبة C القياسية:

جدول 2.4 بعض ملفات الرأس في مكتبةC القياسية

الملـــف	الوصـــــف
<assert.h>	يعلن عن الدالة () assert
<ctype.h>	يعلن عن الدوال لإختبار الحروف
<float.h>	يعلن عن الثوابت التي لها علاقة بالأعداد الحقيقية
<limits.h>	يعرف القيم العظمى والصغرى للأعداد الصحيحة
<math.h>	يعلن عن الدوال الحسابية
<stdio.h>	يعلن عن دوال الدخل والخرج القياسية
<stdlib.h>	يعلن عن دوال المرافق أو الأدوات
<string.h>	يعلن عن الدوال التي تستعمل سلاسل الأحرف
<time.h>	يعلن عن دوال الوقت والتاريخ

هذه ملفات لغة C القياسية . وتستخدم بنفس الطريقة التي تستخدم بها ملفات لغة ++C القياسية مثل <iostream.h> . على سبيل المثال إذا أردت إستخدام دالة الأرقام العشوائية () rand من الملف <stdlib.h> ضع توجيه المعالج الأولي التالي في بداية برنامجك الرئيسي :

```
#include <stdlib.h>
```

مكتبة لغة C القياسية موضحة بتفاصيل أكثر في الفصل الرابع عشر.

2.4 الدوال المعرفة بالمستخدم (المبتكرة)

التنوع الكبير في الدوال الموجودة في مكتبات لغة C ولغة ++C مازال غير كاف لمعظم مهام البرمجة . يحتاج المبرمجين أيضاً لتعريف دوال خاصة بهم.

مثال 3.4 دالة التكعيب

هذا مثال بسيط لدالة معرفة بالمستخدم :

```
//  Returns the cube of the given integer:
int cube  (int  x)
{
        return  x*x*x;
}
```

هذه الدالة ترجع مكعب العدد الصحيح الذي يرسل إليها . لذلك cube (2) سوف يكون 8 . الدالة المبتكرة لها جزئين : الإعلان عن الدالة header وجسم الدالة . الإعلان عن الدالة يحدد نوع القيمة الناتجة من الدالة واسم الدالة وقائمة المعاملات . في المثال 3.4 نوع القيمة الناتجة من الدالة هو int واسم الدالة هو cube وقائمة البارمترات هي int x . لذلك فإن الإعلان عن الدالة cube هو

```
int  cube (int x)
```

جسم الدالة هو البلوك الذي يحتوي على أوامر الدالة والذي يتبع الإعلان عنها . جسم الدالة يحتوي على الأوامر التي تؤدي وظيفة الدالة متضمناً الأمر return لإرجاع القيمة إلى المكان الذي تم استدعاء الدالة منه . جسم الدالة cube هو :

```
{
        return  x*x*x;
}
```

هذا هو جسم لدالة مبسطة . عادة جسم الدالة يكون أكبر كثيراً ، لكن الإعلان عن الدالة يكون على سطر واحد.

أمر رجوع الدالة return له هدفين : ينهي الدالة ويرجع قيمة إلى البرنامج الذي تم منه استعداء الدالة . ويكون شكله كالتالي :

```
return expression;
```

حيث expression هو أي تعبير له قيمة يمكن أن تخصص لمتغير نوعه نفس نوع القيمة الناتجة من الدالة

3.4 برامج الاختبار

عندما تبني الدالة الخاصة بك يجب أن تختبرها مباشر ببرنامج بسيط . هذا البرنامج يسمى برنامج الإختبار test driver للدالة . الغرض منه هو إختبار الدالة وبعد ذلك يتم إهمال هذا البرنامج.

مثال 4.4 برنامج إختبار الدالة () cube.

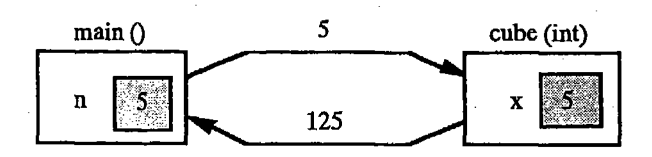
هذا برنامج كامل يتكون من الدالة cube متبوعة ببرنامج إختبار الدالة :

```
//      Returns the cube of the given integer:
int cube (int  x)
{
     return  x*x*x;
}
//   Test driver for the cube function:
main ()
{
     int  n = 1;
     while  (n  != 0)  {
          cin  >>  n;
          cout  <<  cube  (n)  <<  endl;
     }
}
```

```
5
125
-6
-216
0
0
```

هذا البرنامج يقرأ أعداد صحيحة ويطبع مكعباتها إلى أن يكون العدد الداخل يساوي صفر . كل عدد صحيح يقرأ يتم إرساله إلى الدالة () cube بالاستدعاء (n) cube . القيمة الناتجة من الدالة تستبدل التعبير (n) cube وعند ذلك تمر إلى الهدف cout . لاحظ أننا حذفنا التوجيه <iostream.h> #include . هذا التوجيه بالطبع مطلوب لكل برنامج يستعمل cin أو cout . وقد تم حذفه من الأمثلة الأخرى لتوفير حيز فقط .

يمكن أن نتصور العلاقة بين الدالة () main والدالة () cube مثل هذا :



الدالــة () main أرسـلت القيـمـة 5 إلـــى الدالـــة () cube والدالـة () cube أرجعت القيـمة 125 إلى الدالة () main . العامل الحقيقي n أرسل بقيمة إلى البارامتر الأساسي x . هذا يعني أن x قد خصصت لها قيمة n عند استدعاء الدالة .

لاحظ أن الدالة () cube معرفة أعلى الدالة () main في هذا المثال . لأن مترجم ++C يجب أن يتعرف على الدالة () cube قبل أن يستخدمها في () main . المثال التالي يبين دالة مبتكرة تسمى () max ترجع أكبر العددين الصحيحين المرسلين إليها . الدالة لها معاملين .

مثال 5.4 برنامج إختبار الدالة () max

هذه دالة لها معاملان . وهي ترجع أكبر القيمتين المرسلتين إليها :

```
// Returns the larger of the two given integers:
int max (int x, int y)
{
    if (x < y) return y;
    else return x;
}
main ()
{
    int m, n;
    do {
        cin >> m >> n;
        cout << max (m,n) << endl;
    } while (m != 0);
}
```

لاحظ أن الدالة لها أكثر من جملة return . أول جملة يتم تنفيذها تنهي الدالة وترجع القيمة المبينة إلي البرنامج الذي استدعى الدالة .

الأمر return مثل الأمر break . فهو أمر قفز ، يقفز خارج الدالة الموجود فيها . مع أنه من المعتاد وجود الأمر return في نهاية الدالة فإنه من الممكن وضعه في أي مكان داخل الدالة مثل أي أمر آخر .

4.4 الإعلان عن الدوال وتعريفها

المثالان الأخيران يوضحان طريقة واحدة لتعريف الدالة في البرنامج : التعريف الكامل للدالة يكون أعلى البرنامج الرئيسي . هذا ترتيب مبسط جداً لإختبار الدالة . ترتيب آخر أكثر شيوعاً هو كتابة رأس الدالة فقط أعلى البرنامج الرئيسي وبعد ذلك كتابة التعريف الكامل للدالة (الرأس والجسم) أسفل البرنامج الرئيسي . هذا موضح في المثال القادم.

في هذا الترتيب سيكون الإعلان عن الدالة مفصولاً عن تعريفها . الإعلان عن الدالة ببساطة هو رأس الدالة متبوعاً بفاصلة منقوطة . تعريف الدالة هو الدالة كاملة : الرأس والجسم . الأعلان عن الدالة يسمى أيضاً نموذج الدالة prototype .

الإعلان عن الدالة مثل الإعلان عن متغير، الغرض منه تزويد المترجم بكل المعلومات التي يحتاج إليها ليترجم كل الملف . المترجم لا يحتاج لمعرفة كيفية عمل الدالة (جسمها) . هو يحتاج فقط لإسم الدالة وعدد ونوع المعاملات ونوع القيمة المرجعة من الدالة وهذه المعلومات موجودة بالضبط في رأس الدالة . أيضاً الإعلان عن الدالة مثل الإعلان عن متغير يجب أن يكون أعلى أي استخدام لاسم الدالة . لكن تعريف الدالة عندما يكتب منفصلاً عن الإعلان فإنه يمكن أن يظهر في أي مكان خارج الدالة () main وعادة يكون بعدها أو في ملف منفصل.

المتغيرات المدونة في قائمة معاملات الدالة تسمى المعاملات الرسمية formal arguments . وهذه متغيرات محلية لها وجود فقط أثناء تنفيذ الدالة. تدوين هذه المتغيرات في قائمة المعاملات يشكل الإعلان عنها . في المثال السابق المعاملات الأساسية x و y.

المتغيرات المدونة في استدعاء الدالة تسمى المعاملات الحقيقية actual parameters أو الأدلة الحقيقية actual arguments . انها متغيرات مثل أي متغيرات أخرى في البرنامج الرئيسي يجب أن يعلن عنها قبل استخدامها عند استدعاء الدالة . في المثال السابق المعاملات الحقيقية هي m و n.

في هذه الأمثلة المعاملات الحقيقية أرسلت بقيم . هذا يعني أن قيم هذه المعاملات خصصت لما يناظرها من المعاملات الرسمية للدالة . لذلك فإنه في المثال السابق قيمة m خصصت لـ x وقيمة n خصصت لـ y . عند الإرسال بالقيمة فإن المعاملات الحقيقية يمكن أن تكون ثوابت أو تعبيرات جبرية . على سبيل المثال يمكن استدعاء الدالة () max كالتالي max (44, 5*m-n) . هذا سوف يخصص القيمة 44 لـ x وقيمة التعبير الجبري 5*m-n لـ y .

مثال 6.4 الدالة () max مع الإعلان المنفصل عن التعريف

هذا البرنامج هو نفس برنامج إختبار الدالة () max السابق . لكن هنا إعلان الدالة يظهر أعلى البرنامج الرئيسي وتعريف الدالة أسفله.

```
int max  (int,  int);
//  Test driver for the max function:
main ()
{
     int  m,   n;
     do  {
          cin  >>  m  >>  n;
          cout  <<  max (m,n)  <<  endl;
     } while  (m  !=  0);
}
// Returns the larger of the two given integers:
int max  (int x,  int y)
{
     if  (x  <  y) return  y;
     else return x;
}
```

لاحظ أن المعاملات الرسمية x و y مدونة في رأس الدالة (كالمعتاد) لكن غير مدونة في الإعلان عن الدالة.

لاحظ أنه في الحقيقة لا يوجد فرق كبير بين الإعلان عن الدالة والإعلان عن متغير خاصة إذا كانت الدالة ليس لها معاملات . على سبيل المثال في البرنامج الذي يتعامل مع الحروف يمكن أن تحتاج إلى متغير إسمه length لتخزين طول سلسلة الحروف. لكن بديل معقول هو أنه يمكن أن يكون عندك دالة لحساب طول سلسلة الحروف في أي مكان تحتاج إليها بدلاً من تخزين وتحديث القيمة . في هذه الحالة الدالة يمكن أن يعلن عنها كالتالي :

```
int  length  ();
```

بينما المتغير يمكن أن يعلن عنه هكذا :

```
int  length;
```

الفرق الوحيد هو أن إعلان الدالة يحتوي على الأقواس ( ) .

5.4 الترجمة المنفصلة

تعريفات الدوال غالباً تترجم في ملفات منفصلة أو مستقلة . على سبيل المثال كل الدوال المعلن عنها في مكتبة لغة C القياسية تم ترجمة كل منها على حدة . سبب وحيد للترجمة المنفصلة هو إخفاء أو ستر المعلومات .

بمعنى أن المعلومات الضرورية لإتمام الترجمة وليست ضرورية للمبرمج يتم إخفاؤها . الخبرة بينت أن إخفاء المعلومات تسهل فهم ونجاح مشروعات البرامج الكبيرة.

مثال 7.4 الدالة () max بالترجمة المنفصلة

هذا المثال يبين طريقة ترجمة الدالة max وبرنامج إختبارها كل على حدة . برنامج إختبار الدالة موجود في ملف سمي test_max. c والدالة موجودة في ملف سمي max,c .

```
test_max.c

int max (int, int);
// Test driver for the max function:
main ()
{
    int m, n;
    do {
        cin >> m >> n;
        cout << max (m,n) << endl;
    } while (m != 0);
}

max.c

// Returns the larger of the two given integers:
int max (int x, int y)
{
    if (x < y) return y;
    else return x;
}
```

الأوامر الحقيقية التي يمكن أن تستخدمها لترجمة هذه الملفات سوياً تعتمد على نوع الحاسب الذي تعمل عليه. في نظام محطات التشغيل UNIX يمكن أن تستخدم الأوامر التالية :

```
$ c++ -c max_c

$ c++ -c test_max.c

$ c++ -o test_max test_max.o max.o

$ test_max
```

(علامة الدولار هنا هي علامة إنتظار الحاسب prompt) . أول أمر يترجم الدالة max وثاني أمر يترجم برنامج إختبار الدالة منفصلاً وثالث أمر يربطهم سوياً لإنتاج برنامج قابل للتنفيذ سمي test_max والذي يمكن أن ينفذ بالأمر الذي على السطر الرابع.

أول ميزة في ترجمة الدوال منفصلة هي أنه يمكن إختبار كل دالة على حدة قبل استدعائها في البرنامج. بمجرد أن تعلم أن الدالة max تعمل كما ينبغي تستطيع أن تنسى كيفية عملها وتخزنها كصندوق مغلق جاهز للاستعمال حينما يطلب . هذا هو كيفية استعمال الدوال التي في مكتبة math .

ميزة أخرى للترجمة المنفصلة هي سهولة استبدال أي دالة بدالة أخرى مكافئة لها . على سبيل المثال لو أنك أردت أن تكتشف أحسن طريقة لحساب العدد الأكبر من بين عددين . في هذه الحالة يمكنك ترجمة وإختبار هذه الدالة وبعد ذلك تربط هذه الدالة مع أي برامج كانت تستخدم النسخة السابقة من الدالة max ().

6.4 المتغيرات المحلية والدوال

المتغير المحلي local variable هو ببساطة متغير أعلن عنه داخل البلوك ويمكن الإستفادة منه داخل البلوك فقط. حيث أن جسم الدالة نفسه هو بلوك فإن المتغيرات التي يعلن عنها داخل الدالة تكون محلية لهذه الدالة وتكون موجودة فقط أثناء تنفيذ الدالة. المعاملات الرسمية تكون ايضا محلية بالنسبة للدالة.

المثالان التاليان يبينان دوال مع متغيرات محلية.

مثال 8.4 دالة المضروب factorial ( )

مضروب العدد الصحيح n هو العدد n! ونحصل عليه بضرب n في كل الأعداد الموجبة التي أقل من n:

$$n! = (n)(n-1)\ldots(3)(2)(1)$$

على سبيل المثال $5! = (5)(4)(3)(2)(1) = 120$

بناء دالة المضروب يكون كالتالي :

```
int factorial (int  n)
{
     if   (n < 0) return 0 ;
     int  f = 1;
     while  (n > 1)
          f * = n -- ;
     return f;
}
```

هذه الدالة لها متغيرين محليين :n و f .المتغير n يعتبر متغير محلي لأنه أعلن عنه في قائمة متغيرات الدالة. المتغير f يعتبر متغير محلي لأنه أعلن عنه داخل جسم الدالة.

البرنامج التالي يختبر دالة المضروب :

```
int factorial (int);
main ( )
{
    for (int i = -1; i < 6; i ++)
        cout << " " << factorial (i);
    cout << endl;
}
```

```
0  1  1  2  6  24  120
```

هذا البرنامج يمكن ان يترجم منفصلا أو يوضع في نفس ملف الدالة ويترجمان سويا

مثال 9.4 دالة التباديل

التباديل Permutation هي ترتيب لعناصر مجموعة محددة. دالة التباديل P(n,k) تعطي عدد التباديل المختلفة لعدد من العناصر k مأخوذة من مجموعة العناصر n .

طريقة وحيدة لحساب هذه الدالة هي بواسطة المعادلة :

$$p\,(n,k) = \frac{n!}{(n-k)!}$$

على سبيل المثال

$$p\,(5,2) = \frac{5!}{(5-2)!} = \frac{5!}{3!} = \frac{120}{6} = 20$$

لذلك فإنه يوجد 20 تباديل مختلفة لعنصرين من مجموعة مكونة من 5 عناصر.

البرنامج التالي يحقق معادلة دالة التبديل :

```
// Returns p(n,k) , the number of permutations of k from n:
int perm (int n, int k)
{
    if (n < 0 || k < 0 || k > n) return 0;
    return factorial (n) / factorial (n-k) ;
}
```

لاحظ أن الشرط (n<0 || k<0 || k>n) يستخدم لمعالجة الحالات التي يكون فيها أي من المعاملات خارج الحد المسموح به . في هذه الحالات ترجع الدالة القيمة صفر لتدل على وجود خطأ في الدخل . هذه القيمة سوف تميز بالبرنامج الذي استدعى الدالة على أنها إشارة خطأ .

البرنامج التالي هو لإختبار الدالة perm () :

```
int perm (int, int);
main ()
{
    for (int i = -1; i < 8; i++) {
        for (int j = -1; j <= i+1; j++)
            cout << " " << perm (i , j);
        cout << endl;
    }
}
```

```
0 0
0 1 0
0 1 1 0
0 1 2 2 0
0 1 3 6 6 0
0 1 4 12 24 24 0
0 1 5 20 60 120 120 0
0 1 6 30 120 360 720 720 0
0 1 7 42 210 840 2520 5040 5040 0
```

7.4 الدوال void

هي دوال لا تنتج قيمة. في لغات البرمجة الأخري مثل هذه الدالة تسمى إجراء procedure أو برنامج فرعي subroutine . في لغة C++ مثل هذه الدالة تعرف ببساطة بوضع الكلمة المفتاحية void مكان نوع القيمة الناتجة من الدالة. النوع يصف مجموعة من القيم ، فعلى سبيل المثال النوع short يصف مجموعة الأعداد الصحيحة من 32768- إلى 32767 . والنوع void يصف المجموعة الخالية . وبالتالي لا يمكن الإعلان عن متغير مع النوع void . الدالة void هي ببساطة واحدة لا ترجع أي قيمة.

مثال 10.4 الدالة printDate ()

هذه الدالة تطبع التاريخ في صورة حروف إذا أعطيتها الشهر واليوم والسنة في صورة رقمية :

```
void printDate (int,   int,   int);
main  ()
{
     int month, day, year;
     do  {
          cin  >>  month  >>  day  >>  year;
          printDate (month,  day,  year);
     }  while  (month  >  0);
}
void printDate (int  m,   int  d,   int  y)
{
     if  (m  <  1  ||  m  >  12  ||  d  <  1  ||  d  >  31  ||  y  <  0)  {
          cout  <<  "Error:  parameter out of range . \n";
          return;
     }
     switch  (m)   {
          case  1 :    cout  <<  "January " ;          break;
          case  2 :    cout  <<  "February " ;         break;
          case  3 :    cout  <<  "March " ;            break;
          case  4 :    cout  <<  "April " ;            break;
          case  5 :    cout  <<  "May " ;              break;
          case  6 :    cout  <<  "June " ;             break;
          case  7 :    cout  <<  "July " ;             break;
          case  8 :    cout  <<  "August " ;           break;
          case  9 :    cout  <<  "September " ;        break;
          case 10 :    cout  <<  "October " ;          break;
          case 11 :    cout  <<  "November " ;         break;
          case 12 :    cout  <<  "October " ;          break;
     }
     cout  <<  d  <<  " , "  <<  y  <<  endl;
}
```

الدالة printDate () لا ترجع قيمة . الغرض منها فقط هو طباعة التاريخ، لذلك فإن نوع القيم الناتجة من الدالة هو void . الدالة تستخدم جملة switch لتطبع الشهر في صورة حرفية وتطبع اليوم والسنة في صورة أعداد صحيحة.

```
12   7    1941
December 7,  1941
5    16   1994
May  16,  1994
0    0    0
Error: parameter out of range.
```

لاحظ أن الدالة ترجع بدون أي شئ إذا كانت المعاملات خارج المدى المسموح به (على سبيل المثال m > 12 أو y < 0) .

حيث أن دالة void لا ترجع قيمة فإنها لا تحتاج أن يكون بها الأمر return . وإذا كان بها الأمر return فإنها تكون في الصورة المبسطة التالية

```
return;
```

بدون أي تعبير يتبع الكلمة المفتاحية return . في هذه الحالة يكون الغرض من الأمر return هو إنهاء الدالة.

الدالة التي لا ترجع قيمة تقوم بأداء فعل معين . تبعاً لذلك فإنه من الأفضل إستخدام جملة فعلية لإسم الدالة. على سبيل المثال الدالة السابقة سميت printDate بدلاً من جملة إسمية مثل date .

8.4 الدوال البولينية

في بعض الحالات من المفيد استعمال دالة لإيجاد قيمة الشرط كما في جملة الأمر if أو جملة الأمر while . مثل هذه الدوال تسمى الدوال البولينية Boolean functions بعد British logician George Boole (1815-1864) .

مثال 11.4 تصنيف الحروف

البرنامج التالي يصنف 128 حرفاً من حروف الأسكي (ASCII)

```
#include <iostream.h>
#include <ctype.h>
// prints the category to which the given character belongs:
void printCharCategory (char c)
{
    cout << "The character [ " << c << " ] is a ";
    if      (isdigit  (c)) cout  << "digit. \n";
    else if (islower (c)) cout  << "lower-case letter. \n";
    else if (isupper (c)) cout  << "capital letter. \n";
    else if (isspace (c)) cout  << "white space character. \n";
    else if (iscntrl  (c)) cout  << "control character. \n";
    else if (ispunct (c))  cout  << "punctuation mark. \n";
    else                   cout  << "Error. \n";
}
main ()
{
    for (int c = 0;  c < 128;  c++)
        printCharCategory (c) ;
}
```

دالـة void المسماة () printCharCategory تستدعى ستة دوال منطقية هي () isdigit و () islower و isupper و () isspace و () iscntrl و () ispunct . كل واحدة من هذه الدوال معرفة مسبقاً في ملف الرأس <ctype.h> . هذه الدوال تستخدم لإختبار نوع الحرف (أي "c type") .

فيما يلي نعرض جزء من خرج البرنامج :

```
The character  [ ] is a control character.
The character  [ ] is a white space character.
The character  [!] is a punctuation mark.
The character  ["] is a punctuation mark.
The character  [#] is a punctuation mark.
```

خرج البرنامج الكامل يحتوي على 128 سطر.

هــذا المـثال يوضـــح عــدة أفـكار جديدة. الفـكرة الرئيسـية هي استخدام الدوال البولينية () isdigit و () islower و () isupper و () isspace و () iscntrl و () ispunct . عـلى سـبيل المثال إستدعاء الدالة

isspace (c) تختبر الحرف c لتحدد إذا كان حرفاً خالياً. (يوجد ستة حروف أماكن خالية : حرف المجال الأفقي الأفقية t\ وحرف الإنتقال إلى سطر جديد n\ وحرف المسافة الرأسية v\ وحرف التغذية f\ وحرف الرجوع r\ وحرف المسافة). إذا كان الحرف c أي من هذه الحروف عند ذلك ترجع الدالة القيمة 1 بمعنى أنه حقيقي "true" وإلا فإنها ترجع صفر بمعنى أنه غير حقيقي "false" . وضع استدعاء الدالة كشرط في الأمر if يسبب تنفيذ أمر الخرج إذا كان فقط الحرف c واحداً من الحروف الستة السابقة.

كل حرف يتم إختباره داخل الدالة printCharCategory () ، ومع أن البرنامج كان يمكن أن يكتب بدون هذه الدالة المنفصلة إلا أن إستعمال هذا التعديل يجعل البرنامج في صورة أجزاء مترابطة هيكلياً. نحن نؤكد هنا على المبدأ العام للبرمجة الذي يوصي بأن تكون كل مهمة في دالة منفصلة .

مثال 12.4 دالة لإختبار الأرقام الصماء prime

البرنامج التالي هو لدالة منطقية لإختبار إذا كان العدد الصحيح أولي (أصم) :

```
//Returns 1 if p is prime, 0 otherwise:
int isPrime (int p)
{
    float sqrtp = sqrt (p);
    if (p < 2) return 0;        // 2 is the first prime
    if (p == 2) return 1;
    if (p % 2 == 0) return 0;           // 2 is the only even prime
    for (int d = 3; d <= sqrtp; d += 2)
        if (p % d == 0) return 0;
    retrun 1;
}
```

هذه الدالة تعمل بالنظر للمقسوم علية d للرقم المعطى n . فهي تختبر قابلية القسمة وذلك بحساب قيمة الشرط (n % d == 0) . هذا الشرط سوف يكون حقيقياً عندما يكون d قاسماً عليه للرقم n (أي n يقبل القسمة على d) . في هذه الحالة الرقم n لا يمكن أن يكون رقم أصم لذلك فإن الدالة ترجع صفراً لأن الرقم غير أصم "false" . إذا إنتهت الحلقة for بدون إيجاد أي قاسم للرقم n فإن الدالة ترجع 1 وذلك لتحقيق الشرط أن الرقم أصم .

نستطيع إيقاف البحث عن المقسوم عليه بمجرد أن تزيد d عن الجذر التربيعي للقيمة n لأنه إذا كانت n هي حاصل ضرب d*a فإن أحد هذين العاملين (d أو a) يجب أن يكون أقل من أو يساوي الجذر التربيعي n. لقد تم تحديد القيمة sqrt كثابت خارج الحلقة حتى يتم حساب الجذر مرة واحدة فقط ، بينما لو أننا استعملنا الشرط d <= sqrt (n) للتحكم في الحلقة for فإنه سيتم حساب قيمة الجذر التربيعي في نهاية كل تكرار.

تتحسن الكفاءة أيضاً إذا تم الاختبار على الأرقام الزوجية (n == 2) أولاً. في هذه الحالة فإنه بمجرد الوصول إلي الحلقة for نحتاج فقط لإختبار المقسوم عليه من الأعداد الفردية . وهذا يتم بزيادة المقسوم عليه بمقدار 2 في كل تكرار .

البرنامج التالي يختبر الدالة () isprime.

```
#include  <iostream.h>
#include  <math.h>                    / /  defines sqrt () used in isprime ()

int  isPrime (int);

main  ()
{
     for  (int  n = 1;   n <  50;  n++)
     if  (isPrime  (n))    cout    <<    n   <<   "    ";
     cout  <<  endl;
}
```

```
2  3  5  7  11  13  17  19  23  29  31  37  41  43  47
```

لاحظ أن الجملة الفعلية استخدمت في تسمية هذه الدالة مثل دوال "c-type" في المثال السابق. الإسم "isprime" يجعل استخدام الدالة أسهل قراءة بالنسبة للإنسان . على سبيل المثال الأمر التالي :

```
if (isPrime (n))   . . .
```

هي غالباً نفس جملة الإنجليزي العادية "if n is prime" .

يجب أن يكون واضحاً أن هذه الدالة ليست مثالية . في البحث عن المقسوم عليه تحتاج فقط لإختبار الأرقام الأولية لأن كل عدد غير أولي هو مضروب وحيد لأرقام صماء . لكن كيف نستعمل الأرقام الصماء فقط لـ d ؟

الإجابة هي تخزين الأرقام الصماء كما نجدها . لكن هذا يحتاج إلى استخدام مصفوفة لذلك سوف ننتظر إلى أن نصل إلى الفصل الخامس لنفعل ذلك .

مثال 13.4 دالة لحساب السنة الكبيسة

السنة الكبيسة هي سنة يكون فيها يوم زيادة (شهر فبراير 29 يوم) يضاف إلى التقويم العادي. معظمنا يعلم أن السنوات الكبيسة هي السنوات التي تقبل القسمة على 4 . على سبيل المثال 1992 و 1996 سنوات كبيسة . معظم الناس لا يعلموا أنه يوجد استثناء لهذه القاعدة : السنوات المئوية (كل مائة عام) ليست سنوات كبيسة . على سبيل المثال السنوات 1800 و 1900 ليست سنوات كبيسة . أكثر من ذلك يوجد استثناء للاستثناء: السنوات المئوية التي تقبل القسمة علي 400 هي سنوات كبيسة . لذلك عام 2000 سوف تكون سنة كبيسة.

البرنامج التالي هو لدالة منطقية لتنفيذ هذا التعريف :

```
// Returns 1 if y is a leap year, 0 otherwise:
int isLeapYear (int y)
{
    return y % 4 == 0 && y % 100 != 0 || y % 400 == 0;
}
```

الشرط المركب `y % 4 == 0 && y % 100 != 0 || y % 400 == 0` سوف يكون صحيحاً عندما تكون y تقبل القسمة على 4 ولكن لا تقبل القسمة على 100 إلا إذا كانت تقبل القسمة أيضاً على 400 . في هذه الحالات ترجع الدالة القيمة 1 وفي كل الحالات الأخرى ترجع صفر.

البرنامج التالي يختبر وينفذ هذه الدالة :

```
int isLeapYear (int);


// Test driver for the isLeapYear function:
main ()
{
    int n;
    do {
        cin >> n;
        if (isLeapYear (n)) cout << n << " is a leap year. \n";
        else cout << n << " is not a leap year. \n" ;
    } while (n > 1);
}
```

1995
1995 is not a leap year.
1996
1996 is a leap year.
1990
1990 is not a leap year.
2000
2000 is a leap year.
0
0 is a leap year.

9.4 دوال الدخل والخرج

تكون الدوال مفيدة خصوصاً في تغليف المهام التي تحتاج إلى تفاصيل معقدة ليس لها صلة وثيقة بالمهمة الأساسية للبرنامج . على سبيل المثال في عملية سجلات الأشخاص ، يمكن أن يكون عندك برنامج يحتاج إلى بيانات عن أعمار المستخدمين . بإبعاد هذه المهمة في دالة منفصلة تستطيع أن تغلف التفاصيل اللازمة لتأكيد صحة البيانات الداخلة بدون إرباك البرنامج الرئيسي . نحن رأينا سابقاً أمثلة لدوال الخرج. الهدف الوحيد للدالة printDate في المثال 10.4 كان لطباعة التاريخ ممثلاً للبارامترات الداخلة. بدلاً من إرسال المعلومات إلى الخلف إلى البرنامج الذي استدعى الدالة فإنها ترسل معلوماتها إلى المخرج القياسي (شاشة الحاسب). دالة الدخل مثل الدالة التي في المثال السابق بدلاً من استقبال بياناتها من خلال المعاملات الخاصة بها فإنها تقرأ هذه البيانات من مدخل قياسي (لوحة المفاتيح).

مثال 14.4 يوضح دالة للدخل . التحكم في الحلقة التكرارية في هذا المثال يتم عن طريق while (1) التي تجعل الحلقة التكرارية لا نهائية : الشرط (1) دائماً صحيح "true" . لكن في الحقيقة يتم التحكم في تكرار الحلقة بجملة return التي لا تنهي الحلقة فقط ولكن أيضاً تنهي الدالة.

مثال 14.4 دالة لقراءة عمر المستخدم

البرنامج التالي هو لدالة مبسطة تسأل عن عمر المستخدم أو المستخدمة وترجع القيمة التي أرسلت إليها (العمر) . هذه الدالة ترفض أي عدد صحيح يدخل لها يكون غير معقول. هذه الدالة تطلب دخلاً باستمرار حتي تستقبل عدداً صحيحاً في الفترة من 1 إلى 120 .

```
int age ()
{
    int n;
    while (1) {
        cout << "How old are you: ";
        cin >> n;
        if (n < 0) cout << " \a\tYour age could not be negative.";
        else if (n > 120)  cout << "\a\tYou could not be over 120.";
        else return n;
        cout << " \n\tTry again. \n";
    }
}
```

بمجرد استقبال دخلاً مقبولاً من cin فإن الدالة تنتهي بالأمر return حيث ترسل هذا الدخل إلى البرنامج الذي استدعاها. إذا كان الدخل غير مقبول (n < 0 or n > 120) فإنه يسمع صفيراً بطباعة الحرف '\a' ويطبع تعليقاً معيناً، ويسأل المستخدم حاول مرة أخرى "try again" .

لاحظ أن هذا مثال لدالة تحتوي على الأمر retrun ليس في نهاية الدالة.

البرنامج التالي هو لإختبار الدالة وجزء من الخرج.

```
// Prompts the user to input her/her age, and returns that value:
int age ();
// Test driver for the age () function:
main ()
{
    int a = age ();
    cout << "\nYou are " << a << " Years old. \n" ;
}
```

```
How old are you : -10
        your age could not be negattive.
        Try again.
How old are you : 200
        Your could not be over 120.
        Try again.
How old are you: 19
        you are 19 years old.
```

لاحظ أن قائمة معاملات الدالة فارغة . لكن بالرغم من أنها لا تحتوي على معاملات للدخل فإن القوسين () يجب أن يكونا موجودين في كل من رأس الدالة وكل نداء لها .

10.4 الإرسال (النقل) بمرجع

إلى الآن كل البارامترات التي رأيناها في الدوال تنقل (ترسل) بالقيمة . هذا يعني أن التعبير المستخدم في نداء الدالة يقدر أولاً ثم تخصص القيمة الناتجة للبارامتر المقابل في قائمة بارامترات الدالة قبل بدء تنفيذ الدالة . على سبيل المثال في نداء الدالة cube (x) إذا كانت قيمة x هي 4 فإن القيمة 4 تنقل إلى المتغير المحلي n قبل بدء تنفيذ الدالة . حيث أن القيمة 4 تستخدم فقط داخل الدالة فإن المتغير x لا يتأثر بالدالة . لذلك فإن المتغير x هو بارامتر قابل للقراءة فقط read-only .

طريقة الإرسال تسمح باستخدام التعبيرات العامة بدلاً من البارامتر الحقيقي في نداء الدالة . على سبيل المثال الدالة cube يمكن أيضاً أن تستدعى بـ cube (3) أو cube (2*x-3) أو cube (2*sqrt(x) – cube (3)). في كل حالة التعبير الذي بين القوسين يقدر بقيمة وحيدة ترسل إلى الدالة.

طريقة الإرسال بالقيمة عادة هي المطلوبة للدوال . فهي تجعل الدالة أكثر إستقلالاً ومحفوظة من أي أخطاء غير مقصودة . على أي حال في بعض الحالات تحتاج الدالة أن تغير قيمة البارامتر المنقول إليها . هذا يمكن عمله بواسطة الإرسال بالمرجع reference .

لنقل البارامتر بالمرجع بدلاً من النقل بالقيمة ببساطة إلحق العلامة & بنوع البارامتر في قائمة بارامترات الدالة . هذا يجعل المتغير المحلي مؤشراً للبارامتر الحقيقي المنقول إليه. لذلك فإن البارامتر الحقيقي قابل للقراءة والكتابة بدلاً من القراءة فقط . عندئذ أي تغيير للمتغير المحلي داخل الدالة سوف يسبب نفس التغيير للبارامتر الحقيقي الذي انتقل إليه.

لاحظ أن البارامترات المنقولة بالقيمة تسمى بارامترات ذات قيمة value parameters والبارامرات المنقولة بالمرجع تسمى معاملات مرجعية reference parameters.

مثال 15.4 الدالة swap ()

هذه الدالة الصغيرة تستخدم بكثرة في ترتيب البيانات :

```
//  Swaps  x  and y so that each ends up with the other's value:
void swap (float&  x,  float&  y)
{
     float temp  =  x;
     x  =  y;
     y  =  temp;
}
```

الهدف الوحيد لهذه الدالة هو تبديل المعاملين المرسلين إليها . يتم هذا بالإعلان عن المتغيرات الأساسية x و y كمتغيرات بالمرجع reference variables : float& y و float& x . معامل المرجع يجعل x و y مرادفات للبارامترات الحقيقية المرسلة إلي الدالة .

البرنامج التالي يختبر الدالة وجزء من الخرج .

```
void swap (float&,  float&);
// Test driver for the swap function:
main ()
{
     float  a  =  27,  b  =  -5.041;
     cout  <<  a   <<  "   "  <<  b  <<  endl;
     swap (a, b);
     cout  <<  a   <<  "   "  <<  b  <<  endl;
}
```

```
27    -5.041
-5.041   27
```

عند استدعاء الدالة swap (a, b) فإنها تخلق مؤشراتها المحلية x و y لذلك فإن x بالنسبة للدالة هو اسم محلي لـ a و y اسم محلي لـ b . عندئذ فإن المتغير المحلي temp أعلن عنه وأخذ قيمة مبدئية هي قيمة المتغير a وخصصت قيمة b للمتغير a وخصصت قيمة temp للمتغير b ، نتيجة لذلك تكون قيمة a النهائية هي -5.041 وقيمة b هي 27.0 .

لاحظ أن إعلان الدالة :

```
void swap (float&,  float&);
```

يتضمن معامل المرجع & لكل بارامتر بمرجع حتى لو أن البارامترات نفسها حذفت. بعض المبرمجين بلغة C++ يكتب معامل المرجع & ملحق ببداية البارامتر هكذا :

```
void swap (float   &x,  float   &y)
```

بدلاً من الحاقه في نهاية النوع كما نفعل . المترجم يقبل float & x و float &x و float &x أو حتى float&x . وإنها غالباً مسألة تذوق.

مثال 16.4 الإرسال بقيمة والإرسال بمرجع

هذا المثال يبين الفرق بين الإرسال بالقيمة والإرسال بمرجع:

```
void f (int  x,  int&  y)   { x  =  88;   y  =  99; }

main ()
{
     int  a  =  22,   b  =  33;
     cout  <<  "a =  "  <<  a  <<,  " , b = "   <<  b  <<  endl;
     f (a, b);
     cout  <<  "a =  "  <<  a  <<,  " , b = "   <<  b  <<  endl;
}
```

```
a  =  22,    b  =  33
a  =  22,    b  =  99
```

نداء الدالة f (a , b) يرسل a بقيمة إلى x ويرسل b بمرجع إلى y . لذلك x هو متغير محلي خصصت له قيمة a وهي 22 بينما y هي اسم مستعار للمتغير b الذي قيمته 33 . الدالة تخصص 88 للمتغير x لكن ليس لها تأثير على a . عندما تخصص الدالة 99 للمتغير y فهي في الحقيقة خصصت 99 للمتغير b . لذلك عندما تنتهي فإن a مازالت لها القيمة الأصلية 22 بينما b لها القيمة الجديدة 99 . البارامتر الحقيقي a يقرأ فقط بينما البارامتر الحقيقي b يقرأ ويكتب.

الجدول التالي يلخص الفرق بين الإرسال بالقيمة والإرسال بمرجع:

جدول 3.4الإرسال بالقيمة مقابل الإرسال بمرجع

الإرسال بالقيمة	الإرسال بمرجع
int x ;	int &x;
البارامتر الأساسي x هو متغير محلي.	البارامتر الأساسي x هو مؤشر محلي.
يكون مطابق للبارامتر الحقيقي.	يكون مرادف للبارامتر الحقيقي.
لا يمكن تغيير البارامتر الحقيقي.	يمكن تغيير البارامتر الحقيقي.
البارامتر الحقيقي يمكن أن يكون ثابت أو متغير أو تعبير جبري.	البارامتر الحقيقي يجب أن يكون متغير.
البارامتر الحقيقي يقرأ فقط.	البارامتر الحقيقي يقرأ ويكتب.

نحتاج إلى البارامترات بمرجع عندما تكون الدالة ترجع أكثر من قيمة. الدالة يمكن أن ترجع قيمة واحدة مباشرة بجملة return . لذلك في حالة رجوع أكثر من قيمة فإن البارامترات بمرجع يمكن أن تقوم بهذا العمل.

مثال 17.4 حساب مساحة ومحيط دائرة

هذه الدالة من خلال بارامتراتها بمرجع يمكن أن ترجع مساحة area و محيط circumference دائرة طول قطرها r :

```
void computeCircle (double& area , double& circumference , double r)
{
     const double PI = 3.141592653589793;
     area = PI*r*r;
     circumference = 2*PI*r;
}
```

فيما يلي برنامج لإختبار الدالة والخرج :

```
void computeCircle (double& , double& , double);
main ()
{
double r, a, c;
cout << "Enter radius: ";
cin >> r ;
computeCircle (a, c, r) ;
cout << "area = " << a << " , circumference = " << c << endl;
}
```

```
Enter radius: 100
area = 31415.9,  circumference = 628.319
```

لاحظ أن بارامترات الخرج area و circumference تم تسجيلهم في بداية قائمة البارامترات على يسار بارامتر الدخل r . هذا هو شكل لغة C القياسي متطابق مع شكل جمل التخصيص : q = p حيث أن المعلومات (القيمة) تمر من المتغير p المقروء فقط الذي على اليمين إلي المتغير q القابل للقراءة والكتابة الموجود على اليسار.

11.4 الانتقال بمرجع ثابت

يوجد سببين جيدين لانتقال البارامتر بمرجع. إذا كانت الدالة يجب أن تغير قيمة البارمتر الحقيقي كما فعلت الدالة () swap عند ذلك فإنه يجب أن تنتقل بمرجع . أيضاً إذا كان البارامتر الحقيقي المنقول إلى الدالة أخذ حيزاً كبير من الذاكرة في التخزين (على سبيل المثال صورة تشغل واحد ميجا بايت) عندئذ فإنه من الأكثر كفاءة الإنتقال بمرجع لمنعها من الازدواجية . على أي حال هذا أيضاً يسمح للدالة بتغيير قيمة (المحتويات) البارامتر الحقيقية. إذا كنت لا ترغب أن تغير الدالة محتوياتها (على سبيل المثال إذا كان الغرض من الدالة هو طباعة الهدف) عند ذلك الإنتقال بمرجع يمكن أن يشكل خطورة . لحسن الحظ لغة ++C تعطينا بديل ثالث : الإنتقال بمرجع ثابت constant reference . أنه يعمل بنفس طريقة الإنتقال بمرجع ما عدا أن الدالة ممنوعة من تغيير قيمة البارامتر . التأثير هو أن الدالة يمكنها الوصول إلى البارامتر الحقيقي بواسطة البارامتر الأساسي لكن قيمة البارامتر الأساسي لا يمكن أن تتغير أثناء تنفيذ الدالة. البارامتر المنقول بقيمة يسمى قراءة فقط "read only" لأنه لا يمكن أن يكتب (أي يغير) محتويات هذا البارامتر.

مثال 18.4 الإرسال بمرجع ثابت

هذا المثال يوضح ثلاثة طرق لإرسال البارامتر إلى الدالة :

```
void f(int x,  int& y,  const  int& z)
{
    x += z;
    y += z;
    cout << "x = " << x << ", y = " << y << ", z = " << z << endl;
}
```

البارامتر الأول a أرسل بقيمة والبارامتر الثاني b ارسل بمرجع والبارمتر الثالث c ارسل بمرجع ثابت:

```
main ()
{
    int a = 22,  b = 33,  c = 44;
    cout << "a = " << a << ",b = " << b << ", c = " << c << endl;
    f (a, b, c);
    cout << "a = " << a << ",b = " << b << ", c = " << c << endl;
}
```

```
a = 22,   b = 33,  c = 44
x = 66,   y = 77,  z = 44
a = 22,   b = 77,  c = 44
```

الدالة غيرت البارامترات الأساسية x و y لكنها لم تستطع أن تغير z . تغيير الدالة للبارامتر x لم يؤثر على البارامتر الحقيقي a لأنه ارسل بقيمة . تغيير الدالة للبارامتر y له نفس التأثير على البارامتر الحقيقي b لأنه ارسل بمرجع. ارسال البارامترات بمرجع ثابت يستخدم غالباً مع الدوال التي تتعامل مع أهداف كبيرة مثل المصفوفات الموضحة في الفصول القادمة. أهداف الأنواع الأساسية (الأعداد الصحيحة والأعداد الحقيقية ... الخ) عادة يتم ارسالها اما بقيمة (إذا كنت لا ترغب في تغيير الدالة لهم) أو بمرجع (إذا كنت ترغب في تغيير الدالة لهم).

12.4 دوال inline

الدوال inline هي دوال تتضمن أعباء إضافية كثيرة. في هذه الحالة يتم استخدام وقت وحيز زيادة لاستدعاء الدالة وإرسال البارامترات إليها وتحديد مكان لتخزين المتغيرات المحلية وتخزين المتغيرات الحالية وموقع لتنفيذ الدالة في البرنامج الرئيسي .. إلخ . في بعض الحالات من الأفضل تجنب كل هذا وذلك بوصف الدالة لتكون inline . هذا يخبر المترجم باستبدال كل نداء للدالة ببرنامج الدالة نفسه. هذه الدالة بالنسبة للمبرمج مثل الدالة العادية ماعدا استخدام الموصف inline .

مثال 19.4 دالة حساب مكعب الاعداد باستخدام دالة inline

هذا هو نفس برنامج الدالة () cube الذي في المثال 1.4 :

```
inline int cube (int n)
{
    return n*n*n;
}
```

الفرق الوحيد هو في الاسم inline في رأس الدالة. المترجم أخبر بأن يستبدل التعبير cube (n) في البرنامج الرئيسي ببرنامج الدالة الحقيقي n*n*n . لذلك لو أن البرنامج الآتي ترجم

```
main ()
{
    cout << cube (4) << endl;
    int x, y;
    cin >> x;
    y = cube (2*x-3) ;
}
```

سوف تكون النتائج كما لو أن البرنامج نفسه أصبح كالتالي :

```
main ()
{
     cout << (4) * (4) * (4) << endl;
     int x,  y;
     cin >> x;
     y = (2*x-3) * (2*x-3) * (2*x-3);
}
```

عندما يستبدل المترجم نداء الدالة inline ببرنامج الدالة الحقيقي نقول أنه فك expands هذه الدالة.

لاحظ أن لغة ++C القياسية لا تتطلب من المترجم فك الدوال inline . هي فقط تنصح المترجم بعمل ذلك. لو أن واحد لم يتبع هذه النصيحة فإنه مازال مقبول بالنسبة لمترجم لغة ++C القياسية . على الجانب الآخر بعض مترجمات لغة ++C القياسية يمكن أن يفك بعض الدوال البسيطة إذا لم يعلن عنها أنها inline.

13.4 المجال Scope

المجال لاسم معين هو جزء البرنامج الذي استخدم هذا الاسم فيه ، والمجال يبدأ من مكان الإعلان عن هذا الاسم. لو أن الإعلان كان داخل دالة (متضمناً دالة () main) فإن المجال يمتد إلى نهاية البلوك الداخلي الذي يحتوي على هذا الإعلان.

البرنامج يمكن أن يحتوي على أهداف متعددة بنفس الاسم مادامت المجالات متداخلة أو منفصلة ، وهذا موضح بالمثال التالي الذي هو تفصيل للمثال 17.2 .

مثال 20.4 المجالات المتداخلة والمتوازية

في هذا المثال () f و () g دوال عامة وأول x هو متغير عام . لذلك فإن مجالهم يشمل كل الملف . هذا يسمى مجال الملف file scope . ثاني x اعلن عنها داخل () main لذلك فإن مجالها محلي أي أنه يمكن الانتفاع بها داخل () main . ثالث x اعلن عنها داخل البلوك الداخلي لذلك فإن مجالها محدود في هذا البلوك.

```
void f () ;                                  // f () is global
void g () ;                                  // g () is global
int x = 11;                                  // this x is global

main ()
{                                            // begin scope of main ()
```

```
    int x = 22;
    {                                              // begin scope of internal block
        int x = 33;
        cout << " In block inside main () : x = " << x << endl ;
    }                                              // end scope of internal block
        cout << "In main () : x = " << x << endl;
        cout << "In main () : ::x = " << :: : x << endl;    //accesses global x
        f ();
        g () ;
}                                                  //end scope of main ()
```

كل مجال x يبطل مجال x السابق لذلك لا يوجد غموض عند الرجوع إلي المميز x . مجال معامل الدقة : : scope resolution operator يستخدم للوصول الي آخر x أبطل مجالها، في هذه الحالة المتغير العام x قيمته 11:

```
void f ( )
{                                                  //begin scope of f ( )
    int x = 44;
    cout << "In f ( ) : x = " << x << endl;
}                                                  // end scope of f ( )
void g ( )
{                                                  // begin scope of g ( )
    cout << "In g ( ) : x = " << x << endl;
}                                                  // end scope of g ( )
```

```
In block inside main () : x = 33
In main () : x = 22
In main () : : :x = 11
In f () : x = 44
In g () x = 11
```

المتغير x الذي اخذ القيمة 44 مجاله محدود في الدالة f ( ) التي هي مناظرة للدالة main ( ) ولكن مجال الدالة f ( ) متداخل في المجال العام لأول x لذلك فإن مجالها يبطل أول x بداخل f ( ) ، في هذا المثال المكان الوحيد الذي لايبطل فيه مجال أول x هو داخل الدالة g ( ).

14.4 زيادة التحميل

لغة ++C تسمح لك بإستخدام نفس الأسم لدوال مختلفة. إذا كانت قوائم بارامترات الدوال مختلفة فإن المترجم يتعامل معهم كدوال مختلفة . حتي يسهل التفريق بينهما فإن قوائم البارامترات يجب أن تحتوي علي عدد مختلف من البارامترات أو يجب أن يكون علي الأقل أحد أنواع البارامترات مختلف في هذه القائمة.

مثال 21.4 زيادة تحميل الدالة ( ) max

في مثال سابق عرضنا الدالة ( ) max بعددين صحيحين . الآن سوف تعرف دالتين آخرين بنفس الإسم ( ) max في نفس البرنامج.

```
int  max (int,  int);

int max (int,  int,  int);

double max (double, double);

main ( )
{
     cout  << max (99,77) << " " << max (55,66,33) << " "
           << max  (3.4, 7.2) << endl;
}
int max (int x,   int y)
{
     return (x > y  ? x :  y);
}
// Returns the maximum of the three given integers:
int max (int  x, int  y, int z)
{
     int  m = (x > y ? x : y);
     return (z > m ?  z : m);
}
//  Returns the maximum of the two given real numbers:
double max (double  x,  double y)
{
     return (x > y  ? x :  y);
}
```

```
99  66   7.2
```

تم تعريف ثلاثة دوال مختلفة بنفس الاسم max . المترجم يختبر قوائم البارامترات لتحديد أي منهما تستخدم في كل نداء . على سبيل المثال أول نداء يرسل عددين صحيحين ints لذلك فإن النسخة التي تحتوي على عددين صحيحين في قائمة بارامتراتها هي التي تستدعي . (لو أن هذه النسخة غير موجودة فإن نظام الحاسب سوف يرقي العددين 99 و 77 من النوع int إلى النوع double 99.0 و 77.0 ويرسلوا إلي النسخة التي تحتوي على عددين من نوع double في قائمة بارامتراتها).

الدوال المحملة overloaded تستخدم كثيراً في لغة ++C ، وسوف تظهر قيمتها أكثر مع استعمال الطبقات classes في الفصل الثامن.

15.4 الدوال () main و () exit

كل برامج ++C تحتاج إلى دالة تسمى () main . البرنامج الكامل مكون من الدالة () main بالإضافة إلى كل الدوال الأخرى التي تستدعى بطريق مباشر أو غير مباشر من الدالة () main . البرنامج يبدأ بالنداء () main.

مع أنه غير مطلوب فإن معظم المترجمات compiles في لغة ++C تتوقع أن يكون للدالة () main نوع int للرجوع . حيث أن هذا النوع المبدئي للرجوع لأي دالة فإنه لا يحتاج إلى وصف . لذلك عادة نكتب

```
main ()
```

بدلاً من

```
int main ()
```

في أي واحدة من الصور السابقة فإن معظم المترجمات تسمح بحذف الأمر return مع أن البعض الآخر يمكن أن يعطى تحذيراً إذا حذفت . إذا وجد الأمر return فإن الدالة يجب أن ترجع عدداً صحيحاً . معظم المبرمجين بلغه ++C يفضلون الإعلان عن () main بدالة void مثل :

```
void main ()
```

هذا مقبول لعظم المترجمات مع أن البعض سوف يعطي تحذيراً وعندئذ يغير () main تلقائياً إلي النوع int . لو أن المترجم قبل () main على أنها دالة void فإن أي أمر return يجب أن يظهر هكذا:

```
return ;
```

في هذه الحالة () main ليس لها نوع للرجوع .

إذا أردت أن تنهي البرنامج من داخل دالة غير دالة () main ، فإنك لا تستطيع عمل ذلك بإستخدام الأمر return ببساطة . جملة return سوف تنهي فقط الدالة الحالية وتعود إلى الدالة الأخرى التي استدعتها. لحسن الحظ يوجد طريقة أخرى لإنهاء البرنامج ويمكن استخدامها من أي مكان داخل أي دالة. يتم ذلك باستخدام دالة () exit المعرفة في الملف <stdlib.h>.

مثال 22.4 استعمال دالة () exit لإنهاء البرنامج

```
#include <iostream.h>
#include <stdlip.h>

double reciprocal (double x)
{
    if (x == 0) exit (1);
    retrun 1.0/x;
}

main ()
{
    double x;
    cin >> x;
    cout << reciprocal (x);
}
```

لو أن المستخدم أدخل صفراً للمتغير x فإن البرنامج سوف ينتهي من داخل الدالة () reciprocal بدون محاولة القسمة عليه.

16.4 المعاملات التلقائية Default Parameters

لغة ++C تسمح أن يكون للدالة عدد متغير من الأدلة arguments ، وهذا يمكن عمله بتزويد قيم تلقائية للمعاملات الاختيارية.

مثال 23.4 البارامترات التلقائية

هذه الدالة تحل معادلة من الدرجة الثالثة $a0 + a1\ x + a2\ x^2 + a3\ \ x^3$. الحل الحقيقي يتم باستخدام خوارزم Horner الذي يجمع الحسابات $a0 + (a1 + (a2 + a3\ x)x)x$ للحصول على كفاءة أعلى :

```
double p(double, double, double = 0, double = 0, double = 0);

main ()
{
    double x = 2.0003;
    cout << "p(x, 7) = " << p(x, 7) << endl;
```

```
    cout << "p(x, 7, 6) = " << p(x, 7, 6) endl;
    cout << "p(x, 7, 6, 5) = " << p(x, 7, 6, 5) endl;
    cout << "p(x, 7, 6, 5, 4) = " << p(x, 7, 6, 5, 4) endl;
}
double p(double x, double a0 , double a1, double a2, double a3)
}
    retrun a0 + (a1 + (a2 + a3*x)*x)*x;
}
```

النداء p (x, a0, a1, a2, a3) يحل معادلة الدرجة الثانية $a0 + a1\,x + a2\,x^2 + a3\,x^3$.

لكن عندما تكون القيم التلقائية لـ a1 و a2 و a3 صفراً فإن الدالة يمكن أن تستدعى p(x, a0) لتحديد ثابت المعادلة $a0^3$ أو p(x, a0, a1, a2) لحل معادلة الدرجة الثانية $a0 + a1\,x + a2\,x^2$.

لاحظ أن القيم التلقائية معطاة في الإعلان عن الدالة.

فيما يلي نعرض خرج البرنامج بعد التنفيذ:

```
p (x, 7) = 7
p (x, 7, 6) = 19.0018
p (x, 7, 6, 5) = 39.0078
p (x, 7, 6, 5, 4) = 71.0222
```

على سبيل المثال النداء p (x, 7, 6, 5) الذي يكافئ النداء p (x, 7, 6, 5, 0) يحل معادلة الدرجة الثانية $7 + 6x + 5x^2$.

في المثال السابق الدالة يمكن أن تستدعي بـ 2 أو 3 أو 4 أو 5 أدلة (معاملات) ، لذلك فإن تأثير السماح بقيم (تلقائية) للبارامترات هو في الحقيقة يسمح بتغيير البارامترات الحقيقية المرسلة إلى الدالة .

إذا كانت بارامترات الدالة لها قيم تلقائية فإنه يجب أن تظهر هذه البارامترات بقيمها التلقائية في قائمة بارامترات الدالة على يمين البارامترات التي ليس لها قيم كالتالي :

```
void f(int a, int b, int c = 4, int d = 7, int e = 3); // ok

void g(int a, int b = 2, int c = 4, int d, int e = 3); //ERROR
```

أسئلة للمراجعة

1.4 ما هي مميزات استعمال الدوال في تحسين البرنامج ؟

2.4 ما الفرق بين الإعلان عن الدالة وتعريفها ؟

3.4 أين يمكن وضع الإعلان عن الدالة ؟

4.4 متى تحتاج الدالة للتوجيه include ؟

5.4 ما هي مميزات وضع تعريف الدالة في ملف منفصل ؟

6.4 ما هي ميزة الترجمة المنفصلة ؟

7.4 ما هي الفروق بين إرسال البارامتر بالقيمة وإرسالها بمرجع ؟

8.4 ما هي الفروق بين إرسال البارامتر بمرجع وإرسالها بمرجع ثابت ؟

9.4 لماذا ينسب البارامتر المرسل بقيمة إلى أنه يقرأ فقط "read-only" ؟

ولماذا ينسب البارامتر المرسل بمؤشر إلى أنه يقرأ ويكتب "read-write" ؟

10.4 ما هو الخطأ في الإعلان التالي :

```
int f(int a, int b = 0, int c);
```

مسائل محلولة

11.4 في المثال 13.4 التعبير التالي كان يستخدم لاختبار إذا كانت y سنة كبيسة:

```
y%4 == 0 && y%100 != 0 || y%400 == 0
```

هذا التعبير ليس في الصورة الأكثر كفاءة . إذا كانت y لا تقبل القسمة على 4 فإن التعبير سوف يظل يختبر الشرط y%400 == 0 الذي هو في الأصل غير صحيح. لغة C++ تنجز القصر "short circuiting" الذي يعني أن الأجزاء التالية من الشرط المركب يتم إختبارها فقط عند الضرورة.

أوجد الشرط المركب المكافئ الذي يكون أكثر كفاءة نتيجة القصر.

الشرط المركب :

```
y%4 == 0 && (y%100 != 0 || y%400 == 0)
```

مكافئ وأكثر كفاءة للشرط السابق . الشرطان من الممكن أن يكونا متكافئان بإختبار قيمهما في الأربع حالات الممثلة بأربع قيم للمتغير y 1995 و 1996 و 1900 و 2000.

هذا الشرط يكون أكثر كفاءة إذ كانت y لا تقبل القسمة على ٤ (الحالة الأكثر احتمالاً) حيث أنه لن يختبر y مرة أخرى .

12.4 صف كيف أن دالة void ببارامتر واحد بمرجع يمكن أن تحول إلى دالة مكافئة non-void ببارامتر واحد بقيمة .

حول البارامتر بمرجع إلى قيمة ترجع من الدالة . على سبيل المثال الدالة

```
void  f (int &n)
{ n  *=  2;
}
```

يكافئ الدالة

```
int  g (int  n)
{ return  2*n;
}
```

هاتان الدالتان مختلفتان في الاستدعاء :

```
int  x  =  22;  y  =  33;
f (x);
y  =  g (y);
```

مسائل برمجة محلولة

13.4 أكتب برنامج بسيط كالذي في مثال ٢.٤ لإختبار المتطابقة $\cos 2x = 2\cos^2 x - 1$.

البرنامج التالي مثل المثال 2.4 :

```
#include  <iostream.h>
#include  <math.h>

main  ()
{
     for  (float  x  =  0;   x  <  1;   x  +=  0.1)
          cout  <<  cos  (2*x)  <<  '\t'  <<  2*cos (x)  *cos (x)  -1  << endl;
}
```

```
1            1
0.980067     0.980067
0.921061     0.921061
0.825336     0.825336
0.696707     0.696707
0.540302     0.540302
0.362358     0.362358
0.169967     0.169967
-0.0291997   -0.0291997
-0.227202    -0.227202
```

كل قيمة في العمود الأول متطابقة مع نظيراتها في العمود الثاني مما يدل على أن التطابق حقيقي في القيم العشرة التي تم اختبارها للمتغير x .

14.4 طريقة أكثر كفاءة لحساب دالة التباديل p(n, k) من المكن أن تكون بالصورة التالية :

$$p(n, k) = (n)(n-1)(n-2) \ldots (n-k+2)(n-k+1)$$

هذا يعني ضرب الأعداد الصحيحة k من n إلى n - k + 1 . استخدم هذه الصيغة لإعادة كتابة واختبار الدالة perm () من المثال 9.4 .

لحساب مضروب k من الأعدد الصحيحة نستخدم الحلقة التكرارية for التي تتكرر k من المرات. في كل مرة p يضرب في العدد n الذي يتناقص في كل مرة . النتيجة هي أن 1 يضرب في n و n-1 و n-2 حتى n- k +1 .

```
int perm (int, int);

main ()
{
     for (int i = -1; i < 8; i++) {
          for (int j = -1; j <= i+1; j++)
               cout << " " << perm (i, j);
     cout << endl;
     }
}
// Returns p(n, k) , the number of permutations of k from n:
int perm (int n, int k)
{
```

```
    if (n < o || k < 0 || k > n) return 0;
    int p = 1;
    for (int i = 1;  i <= k; i++, n--)
        p  *= n;
    return p;
}
```

نتيجة خرج البرنامج هي نفس الخرج في المثال 9.4 :

```
0  0
0  1  0
0  1  1  0
0  1  2  2  0
0  1  3  6  6  0
0  1  4  12  24  24  0
0  1  5  20  60  120  120  0
0  1  6  30  120  360  360  720  0
0  1  7  42  210  840  2520  5040  5040  0
```

15.4 الدالة التوافقية c (n, k) تعطي عدد المجموعات الفرعية المختلفة (الغير مرتبة) الموجودة في مجموعة. مكونة من عدد n من العناصر حيث كل مجموعة من هذه المجموعات الفرعية مكونة من k من العناصر.

هذه الدالة يمكن حسابها من الصيغة :

$$c(n,k) = \frac{n!}{k!\,(n-k)!}$$

نفذ هذه الصيغة .

هذا هو تنفيذ مباشر لهذه الصيغة :

```
int comb (int, int);
main ()
{
    for (int i = -1; i < 8; i++) {
        for (int j = -1; j <= i+1; j++)
            cout << " " << comb (i, j);
        cout << endl;
    }
}
```

```
int factorial (int);
// Returns C (n, k), the number of combinations of k from n:
int comb (int n, int k)
{
    if (n < 0 || k < 0 || k > n) return 0;
    return factorial (n) / (factorial (k) * factorial (n-k) );
}
```

```
0  0
0  1  0
0  1  1  0
0  1  2  1  0
0  1  3  3  1  0
0  1  4  6  4  1  0
0  1  5  10  10  5  1  0
0  1  6  15  20  15  6  1  0
0  1  7  21  35  35  21  7  1  0
```

لاحظ أن الدالة () factorial يجب أن يعلن عنها فوق الدالة () comb لأنها تستخدم بهذه الدالة . لكن لا تحتاج أن يعلن عنها فوق الدالة () main لأنها تستخدم فيها .

16.4 أكتب وإختبر الدالة () digit :

```
int digit (int n, int k)
```

هذه الدالة ترجع الخانة رقم k (k^{th} digit) من العدد الصحيح n . على سبيل المثال إذا كان العدد الصحيح n هو 29415 فإن النداء (digit (n, 0 سوف يرجع الرقم 5 والنداء (digit (n, 2 سوف يرجع الرقم 4 . لاحظ أن الخانات مرقمة من اليمين إلى اليسار بالخانة رقم صفر.

هذا يحذف الخانة التي في أقصى اليمين للعدد n بمقدار k من المرات . وهذا ينقص n إلى عدد صحيح خانته التي في أقصى اليمين هي نفس الخانة رقم k للعدد الصحيح الأصلي . هذه الخانة سنحصل عليها من باقي خارج القسمة على 10 .

```
int digit (int,  int);

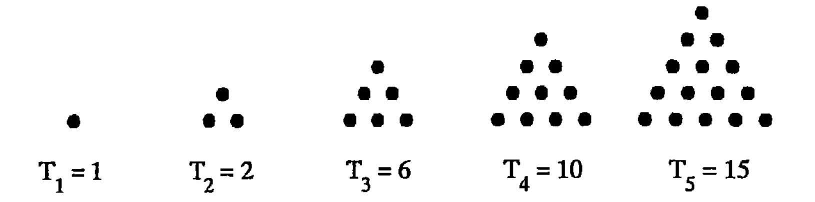
main ()
{
     int  n,  k;
     cout   <<  " Integer: ";
     cin   >>  n;
     do  {
          cout   <<  "Digit: ";
          cin  >>  k;
          cout  <<  "The  "  <<  k  <<  "  th digit of "  <<  n  <<  " is "
               <<  digit  (n, k)  <<  endl;
     }  while  (k  >  0) ;
}

// Returns the kth digit of the integer n :
int digit  (int  n,  int  k)
{
     for  (int  i  =  0;  i  <  k;  i ++)
          n  /=  10;  //  remove right-most digit
     return  n% 10;
}
```

```
Integer :   123456789
Digit :  8
The 8th digit of 123456789 is  1
Digit :  4
The 4th digit of 123456789 is  5
Digit :  1
The 1th digit of 123456789 is  8
Digit :  0
The 0th digit of 123456789 is  9
```

تنفيذ هذا البرنامج كان على جهاز حاسب فيه النوع int يشغل 9 خانات .

17.4 اللغة اليونانية القديمة قسمت الأرقام هندسياً . على سبيل المثال الرقم كان يسمى "مثلث" إذا كان عدد حصوات هذا الرقم يمكن ترتيبها في شكل مثلث متماثل . أول ثمانية أرقام للمثلثات هي 1 و 3 و 6 و 10 و 15 و 21 و 28 و 36 :



أكتب واختبر الدالة البولينية

```
int isTriangular (int  n)
```

حيث ترجع هذه الدالة 1 إذا كان العدد الصحيح المعطى n هو عدد مثلثي وإلا فإنها ترجع صفراً. المعامل n يكون مثلثي فقط إذا كان هو مجموع الأعداد المتتالية .... + 3 + 2 + 1 . لذلك نحن يجب أن نحسب مجاميع الأعداد المتتالية إلى أن نجد أحد هذه المجاميع أكبر من أو يساوي n . إذا كان هذا المجموع يساوي n عند ذلك تكون n عدد مثلثي وإلا فهي ليست عدد مثلثي :

```
int isTriangular (int);
main ()
{
     int  n;
     do {
          cin >> n;
          if  (isTriangular (n) )  cout << n << " is triangular. \n";
          else cout << n << is not triangular. \n";
     } while (n > 0);
}
// Returns 1 if n is a triangular number (1, 3, 6, 10, 15, etc.):
int isTriangular (int  n)
{
     int i = 0,  sum = 0;
     while (sum < n)
          sum += ++i;
     if (sum == n)  return 1;
     else return 0;
}
```

```
10
10 is triangular.
8
8 is not triangular.
6
6 is triangular.
2
2 is not triangular.
1
1 is triangular.
0
0 is triangular.
```

18.4 أكتب دالة لحساب القيمة العظمى من بين ثلاثة أعداد صحيحة بحيث تستخدم هذه الدالة دالة القيمة العظمى لعددين صحيحين .

نفترض أن الدالة (int, int) max موجودة مسبقاً :

```
int max (int, int);
int max (int x, int y, int z)
{
    int max (int, int);
    return max (max (x, y) , z);
}
```

19.4 أكتب برنامجاً يحول الاحداثيات المتعامدة إلى الاحداثيات القطبية.

كل نقطة في مستوى الاحداثيات لها زوج وحيد (y , x) في الاحداثيات المتعامدة وزوج وحيد (r, θ) في الاحداثيات القطبية

$0 \le \theta \le 2\pi$ و $r \ge 0$

الدالة التالية تحول من الإحداثيات المتعامدة إلى الاحداثيات القطبية . حيث أن الخرج يتكون من أكثر من متغير واحد فإن متغيرات الخرج r و t ترسل بمرجع:

```
void rectangularTopolar (double& r, double& t, double x, double y)
{
    const double PI = 3.14159265358979323 85;
```

```
        r = sqrt (x*x + y*y) ;
        if (x > 0)
            if (y >= 0) t = atan (y/x);
            else t = atan (y/x) + 2*PI;
        else if (x == 0)
            if (y > 0) t = PI/2;
            else if (y == 0) t = 0;
            else t = 3*PI/2;
        else t = atan (y/x) + PI;
}
```

20.4 أكتب برنامج لمحاكاة لعبة القمار .

لعبة القمار تلعب بإثنين من زهر الطاولة . في كل مرة يتم قذف زهرا الطاولة ويستخدم مجموع الأرقام الموجود على الزهرين في تحديد الفائز . المجموع سوف يكون عدداً صحيحاً في المدى من 2 إلى 12 حيث أن أوجه كل زهر مرقمة من 1 إلى 6 . اللاعب يكسب إذا ألقى الزهرين وكان مجموع الأرقام 7 أو 11 ، ويخسر إذا كان مجموع الأرقام 2 أو 3 أو 12 . إذا كان مجموع الأرقام 4 أو 5 أو 6 أو 8 أو 9 أو 10 فإن هذا الرقم يصبح نقطة لصالحه . عند ذلك يكرر إلقاء الزهر إلى أن يكسب بالنقط أو يخسر بحصوله على العدد 7 .

```
#include <iostream.h>
#include <stdlib.h>
#include <time.h>

void initializeSeed ();
int toss ();
void win ();
void lose ();

main ()
{
    initializeSeed ();
    int point = toss ();
    if (point == 2 || point == 3 || point == 12) lose ();
    if (point == 7 || point == 11 ) win ();
```

```
    int t;
    for (;;) {
        t = toss ();
        if (t == 7) lose ();
        if (t == point) win ();
    }
}

void initializeSeed ()
{
    unsigned seed = time (NULL);
    srand (seed) ;
}
int toss ()
{
    int die1 = rand () /10%6 + 1;
    int die2 = rand () /10%6 + 1;
    int t = die1 + die2;
    cout << "\tYou tossed a " << t << endl;
    return t;
}

void win ()
{
    cout << "\tYou won. \n";
    exit (0);
}

void lose ()
{
cout << "\tYou lost. \n" ;
exit (0);
}
```

```
You tossed a 4
You tossed a 6
You tossed a 7
You lost.
```

```
You tossed  a 8
You tossed  a 3
You tossed  a 6
You tossed  a 3
You tossed  a 8
You won.

You tossed  a 7
You won.

You tossed  a 5
You tossed  a 8
You tossed  a 2
You tossed  a 3
You tossed  a 11
You tossed  a 9
You tossed  a 8
You tossed  a 7
You lost.

You tossed  a 12
You lost.
```

مسائل برمجة إضافية

دوال مكتبة C القياسية

21.4 أكتب برنامجاً مبسطاً كالذي في المثال 2.4 لاختبار المتطابقة $\cos^2 x + \sin^2 x = 1$.

22.4 أكتب برنامجاً مبسطاً كالذي في المثال 2.4 لاختبار المتطابقة

$\tan 2x = 2\tan x / (1 - \tan^2 x);$

23.4 أكتب برنامجاً مبسطاً كالذي في المثال 2.4 لاختبار المتطابقة

$\cosh^2 x - \sinh^2 x = 1$

24.4 أكتب برنامجاً مبسطاً كالذي في المثال 2.4 لاختبار المتطابقة

$a \sin x + a \cos x = \pi/2$

25.4 أكتب برنامجاً مبسطاً كالذي في المثال 2.4 لاختبار المتطابقة

$$\log x^2 = 2\log x$$

26.4 أكتب برنامجاً مبسطاً كالذي في المثال 2.4 لاختبار المتطابقة

$$b^x = e^{(x \log b)}$$

27.4 أكتب برنامجاً مبسطاً لاختبار الدوال الموجودة في الجدول 1.4

الدوال المبتكرة

28.4 أكتب واختبر الدالة () area التالية التي ترجع مساحة دائرة إذا أعطى لها قطر الدائرة r :

```
float area (float r)
```

29.4 أكتب واختبر الدالة () min التالية التي ترجع أصغر العددين المعطيين لها

```
int min (int x , int y)
```

30.4 أكتب واختبر الدالة () min التالية التي ترجع أصغر الثلاثة أعداد المعطاة لها.

```
int min (int x, int y, int z)
```

31.4 أكتب واختبر الدالة () min التالية التي ترجع أصغر الأربع أعداد المعطاة لها .

```
int min (int x, int y, int z, int w)
```

32.4 أكتب واختبر الدالة () min التالية التي ترجع العدد الأصغر في الثلاثة المعطاة لها والتي تستخدم الدالة min (int , int) لإيجاد ورجوع أصغر الثلاثة أعداد المعطاة لها.

```
int min (int x , int y , int z )
```

33.4 أكتب واختبر الدالة () min التالية والتي ترجع العدد الأصغر في الأربعة المعطاة لها والتي تستخدم الدالة min (int, int) لإيجاد ورجوع أصغر الأربعة أعداد المعطاة لها.

```
int min (int x, int y, int z, int w)
```

34.4 أكتب واختبر الدالة () min التالية التي تستخدم الدالة min (int, int, int) لإيجاد ورجوع أصغر الأعداد الأربعة المعطاة لها

```
int min (int x, int y, int z , int w)
```

35.4 أكتب واختبر الدالة () power التالية التي ترجع x^p حيث p أي عدد صحيح غير سالب :

float power (float x, unsigned p)

36.4 نفذ الدالة () factorial التي تستخدم الحلقة التكرارية for . حدد أي قيم للمتغير n سوف تسبب فائض حسابي لـ factorial (n).

37.4 الدالة المركبة c(n, k) يمكن حسابها من الصيغة التالية :

$$c\,(n,\ k)\ =\ \frac{p(n,\ k)}{k!}$$

استخدم هذه الصيغة في اعادة كتابة واختبار الدالة () comb التي في المسألة 15.4.

38.4 الصيغة التالية هي طريقة أكثر كفاءة لحساب c(n, k)

$$c(n, k) = (n/1)\ ((n-1)/2)\ ((n-3)/3)\ \ldots.\ \ ((n-k + 2)/(k-1))\ ((n-k +1)/k)$$

هذه الطريقة تتناوب القسمة والضرب . استخدم هذه الصيغة لاعادة كتابة واختبار الدالة () comb الموجودة في المسألة 15.4 تلميح : استخدم الحلقة التكرارية for مثل التي في المسألة 14.4.

39.4 مثلث Pascal هو مثلث مجموعة أعداد مرتبة كالتالي :

```
                1
              1   1
            1   2   1
          1   3   3   1
        1   4   6   4   1
      1   5   10  10  5   1
    1   6   15  20  15  6   1
  1   7   21  35  35  21  7   1
1   8   28  56  70  56  28  8   1
```

كل رقم في مثلث pascal هو واحد من التركيبات c(n, k) . (انظر المسألة 15.4 والمسألة 38.4) . لو أننا حسبنا عدد الصفوف والأعمدة القطرية بدءاً من الصفر ، فإن الرقم في الصف n والعمود k يكون c(n, k) . على سبيل المثال الرقم في الصف 6 والعمود 2 هو 15 = (2 ,6)c . أكتب برنامجاً يستخدم الدالة () comb ليطبع مثلث pascal من أعلى إلى أسفل إلى الصف رقم 12.

40.4 أكتب واختبر دالة لتنفيذ خوارزم Euclidean وترجع القاسم المشترك الأكبر من بين العددين الصحيحين الموجبين المعطيين للدالة . (انظر المثال 19.3) .

41.4 اكتب واختبر دالة تستخدم دالة القاسم المشترك الأكبر (مسألة 40.4) لترجع أقل مضاعف مشترك للعددين الصحيحين الموجبين المعطيين للدالة.

الدوال البولينية

42.4 اكتب واختبر الدالة () issquare التالية التي تحدد إذا كان العدد الصحيح المعطى لها هو عدد تربيعي أم لا :

```
int issquare (int n)
```

أول عشرة أرقام مربعة هي 1 و 4 و 9 و 16 و 25 و 36 و 49 و 64 و 81 و 100.

43.4 اكتب واختبر الدالة () ispentagonal التالية التي تحدد إذا كان العدد الصحيح المعطى لها هو عدد خماسي أم لا:

```
int ispentagonal (int n)
```

أول عشرة أرقام خماسية هي 1 و 5 و 12 و 22 و 35 و 51 و 70 و 92 و 117 و 145 .

زيادة التحميل

44.4 اكتب واختبر الدالة () drawSquare التي تطبع مربع عرضه w باستخدام حرف النجمة * .

```
void drawSquare (int w)
```

45.4 اكتب واختبر الدالة () drawRectangle التالية التي تطبع مستطيل قاعدته w وارتفاعه h باستخدام حرف النجمة * .

```
void drawRectangle (int w, int h)
```

46.4 اكتب واختبر الدالة () average التالية التي ترجع متوسط أربعة أعداد :

```
float average (float x1, float x2, float x3, float x4)
```

47.4 اكتب واختبر الدالة () average التي ترجع متوسط أعداد صحيحة موجبة قد تصل إلى أربعة:

```
float average (float x1, float x2 = 0, float x3 = 0, float x4 = 0)
```

الارسال بمرجع

48.4 اكتب واختبر الدالة () computeCircle التالية التي ترجع المساحة a والمحيط c لدائرة نصف قطرها r :

```
void computeCircle (float& a, float& c, float r)
```

49.4 اكتب واختبر الدالة () computeRectangle التالية التي ترجع المساحة a والمحيط p لمستطيل عرضه w وارتفاعه h :

```
void computeRectangle (float& a, float& p, float w, float h)
```

50.4 اكتب واختبر الدالة () computeTriangle التالية التي ترجع المساحة a والمحيط p لمثلث أطوال أضلاعه a و b و c :

```
void computeTriangle (float& a, float& p, float a, float b, float c)
```

51.4 اكتب واختبر الدالة () computeSphere التي ترجع الحجم V ومساحة السطح S لكرة نصف قطرها r:

```
void comuteSphere (float& v, float& s, float& r)
```

52.4 اكتب واختبر الدالة () computeCylinder التالية التي ترجع الحجم V ومساحة السطح S لاسطوانة نصف قطرها r وارتفاعها h :

```
void computeCylinder (float& v, float s, float r, float h)
```

53.4 اكتب واختبر الدالة () computeMeans التالية التي ترجع المتوسط الحسابي a والوسط الهندسي g والوسط التوافقي h لثلاثة أعداد موجبة :

```
void computeMeans (float& a, float& g, float& h, float x1,
                   float x2 = 0, float x3 = 0)
```

الدوال التي تحتوي على أدلة لها قيم تلقائية

54.4 اكتب واختبر الدالة () polynomial كالتي في المثال 23.4 والتي تحل متعددة الحدود إلى الدرجة السادسة (أي أن أكبر أس لـ x هو x^6)

55.4 أكتب واختبر الدالة المسماة () content والتي ترجع إما طول الفترة [x1, x2] أو مساحة مستطيل [x1, x2] x [y1, y2] أو حجم متوازي السطوح [x1, x2] x [y1, y2] x [z1, z2] تبعاً لعدد البارامترات المنقولة إلى الدالة 2 أو 4 أو 6 . على سبيل المثال نداء الدالة بأربعة بارامترات

content (3.0, 8.0, -4.0, 6.0) سوف ترجع (3.0 - 8.0)((4.0-) - 6.0) = 50.0

56.4 اكتب واختبر الدالة المسماة () dotproduct التي ترجع إما ضرب رقمين x1 و y1 أو الضرب القياسي لمتجهين (x1, y1) و (x2, y2) أو الضرب القياسي لمتجهين ثلاثيا الأبعاد (x1, y1, z1) و (x2, y2, z2) وذلك تبعاً لعدد البارامترات المنقولة إلى الدالة إما 2 أو 4 أو 6 . على سبيل المثال نداء الدالة بأربعة بارامترات dotproduct (3.0, 8.0, -4.0, 6.0) سوف يرجع 36.6 = (6.0)(8.0) + (4.0-) (3.0)

دوال تستدعي دوال اخرى

57.4 اكتب واختبر الدالة max التالية التي تستخدم الدالة max (int, int) لحساب وارجاع أكبر الأعداد الصحيحة الأربعة المعطاة :

```
int  max (int x,  int y,  int z,  int w)
```

58.4 اكتب واختبر الدالة min التالية التي تستخدم الدالة min (int , int , int) لحساب وارجاع أصغر الأعداد الأربعة المعطاة :

```
int  min (int x,  int y,  int z,  int w)
```

التعديل

59.4 عدّل برنامج Monte Carlo (المسألة 21.3) لحساب قيمة π بحيث يكون في صورة وحدة module .

60.4 عدّل برنامج Monty Hall (إنظر المسألة 22.3 والمسألة 61.3) بحيث أن main () يكون مجموعة نداءات للدوال :

```
main ()
{
      printIntroduction () ;
      initializeSeed () ;
      int  car,  choice , open , option;
      car  =  randomInteger  (1, 3);
      get  (choice);
      set  (open,  option,  car,  choice);
      if  (change (open,  option))  choice = option;
      printResults (car,  choice);
}
```

61.4 عدّل برنامج monte Hall (المسألة 60.4) بحيث أنه يلعب اللعبة 6000 مرة . لا تستخدم طريقة "switch" في الثلاثة آلاف مرة الأولى واستخدم طريقة "switch" في الثلاثة آلاف مرة الثانية . تتبع في كل طريقة بأي جزء يكسب اللاعب واطبع النتائج.

62.4 عدّل برنامج لعبة القمار (المسألة 22.3) بحيث أنه يلعب اللعبة 3600 مرة . واطبع عدد ونسبة الفوز .

اجابات لأسئلة المراجعة

1.4 الدالة التي ترجمت على انفراد يمكن أن ينظر إليها كصندوق مغلق مستقل يؤدي مهمة معينة . بمجرد أن يتم الاختبار الشامل للدالة فإن المبرمج لا يحتاج لمعرفة كيفية عمل الدالة . هذا يجعل المبرمج يركز في بناء البرنامج الرئيسي. أكثر من ذلك لو أن طريقة أفضل وجدت أخيراً لبناء الدالة فإنه يمكن استبدال النسخة السابقة من الدالة بدون التأثير على البرنامج الرئيسي.

2.4 الإعلان عن الدالة (يسمى أيضاً نموذج أولي) وهو ضروري فقط في رأس الدالة. تعريف الدالة هو الدالة كاملة: رأس الدالة وجسمها. الإعلان يعطي فقط المعلومات التي نحتاج إليها في نداء الدالة : إسمها وأنواع البارامترات ونوع القيمة المرتجعة. والإعلان هو المواجهة بين الدالة والمنادي عليها . التعريف يعطي كل المعلومات عن الدالة بما فيها التفاصيل عن كيفية عملها والتعريف أيضاً هو بناء الدالة .

3.4 الدالة يمكن أن يعلن عنها في أي مكان مادام الإعلان عنها يكون فوق أي إشارة لها. لذلك يجب أن يأتي الإعلان قبل أي نداء لها وإذا كان تعريف الدالة منفصل فإنه يجب أن يأتي بعد الإعلان عنها .

4.4 التوجيه include يستخدم لضم ملفات أخرى. الإعلان عن الدالة وتعريفها موجودين في ملف مستقل "ملف رأس" (بالامتداد h.) إذا كانت الإعلانات فقط موجودة في ملف رأس عندئذ فإن التعريفات يجب أن تترجم في ملفات أخرى مستقلة.

5.4 ميزة وضع تعريف الدالة في ملف رأس مستقل هي أنها لا تكون موجودة في المحرر عندما تحدث تغييرات للدوال التي تناديها.

6.4 ميزة الترجمة المستقلة للدالة هي أنها لا تحتاج إلى اعادة ترجمة عند اعادة ترجمة الدوال التي تناديها.

7.4 ارسال البارامتر بالقيمة هي نسخة من البارامتر الحقيقية المناظرة لها.

إرسال البارامتر بمرجع هي ببساطة اعادة تسمية للبارامتر الحقيقية المناظرة لها .

8.4 ارسال البارامتر بمرجع ثابت لا يمكن تغييره بالدالة المرسل إليها .

9.4 البارامتر المرسل بقيمة لا يمكن تغييره (اعادة كتابته).

10.4 الدالة لها قيمة تلقائية default للبارامتر (b) الذي يسبق البارامتر (c) الذي ليس له قيمة تلقائية . إن هذا يخالف الشرط الأساسي وهو أن البارامترات التي لها قيم تلقائية تكون موجودة في آخر قائمة بارامتر الدوال.

الفصل الخامس

5

الصفــــــوف

Arrays

1.5 مقدمـــة :

الصف هو عبارة عن تتابع من الأهداف كلها من نفس النوع . هذه الأهداف تسمى عناصر الصف ويتم ترقيمها بالتتابع 0 ، 1 ، 2 ، 3 ، .... هذه الأرقام تسمى الفهرس index أو القيم الجانبية subscripts للصف. إن تعبير القيم الجانبية يتم استخدامه لأنه كتتابع حسابي يمكن كتابته كالتالي : a0 ، a1 ، a2 ، .... هـذه الأرقام الجانبية تحدد مكان العنصر في الصف ، وعلى ذلك فإنها تحقق الاتصال المباشر بالصف .

إذا كـان اسم الصف هو a ، فإن a [0] هو اسم العنصر الموجود في المكان رقم صفر (أول مكان) ، و a [1] هو اسم العنصر الموجود في المكان رقم 1 ، وهكذا . وعامة فإن العنصر i هو العنصر الموجود في المكان رقم i-1 ، وعلى ذلك فإنه إذا كان الصف يحتوي عدد n من العناصر فإن أسماء هذه العناصر ستكون a[0] ، a[1] ، a[2] ، .... ، a[n-1] . يمكن تصور الصف كالتالي :

a	11.11	33.33	55.55	77.77	99.99
	0	1	2	3	4

هذا الرسم يبين صف اسمه a يتكون من خمسة عناصر : العنصر الأول a[0] يحتوي 11.11 ، والعنصر a[1] يحتوي 33.33 ، والعنصر a[2] يحتوي 55.55 ، والعنصر a[3] يحتوي 77.77 ، والعنصر a[4] يحتوي 99.99 . هذا الشكل يمثل في الحقيقة جزء من الذاكرة الخاصة بالحاسب لأن أي صف يتم تخزينه عادة بهذه الطريقة بحيث تكون كل عناصره في تتابع حقيقي. طريقة ترقيم العنصر i بالرقم i-1 تسمي بطريقة الفهرسة ذات القاعدة صفر . استخدام هذه الطريقة للفهرسة يضمن أن رقم أي عنصر يكون مساوي لعدد الخطوات التي يبعدها هذ العنصر من العنصر الأول a[0] . فمثلاً العنصر a[3] يبعد ثلاث خطوات من العنصر a[0] . مميزات هذه الطريقة سيتم توضيحها في الفصل السادس عندما نرى العلاقة بين الصفوف والمؤشرات .

2.5 معالجة الصفوف

ظاهرياً جميع البرامج المفيدة تستخدم الصفوف . من الأسباب التي تجعل الصفوف لها هذه الفائدة هي إمكانية السماح لاسم واحد بفهرس متغير أن يستخدم بدلاً من أسماء متعددة، وهذا يجعل من السهل عمل أشياء كثيرة كان من الصعب جداً تحقيقها بدون استخدام الصفوف .

مثال 1.5 طباعة تتابع مرتب

هذا البرنامج يقرأ 4 أرقام ثم يقوم بطباعتهم بترتيب عكسي لعملية قراءتهم :

```
main ()
{
     double  a [4] ;
     cout  << " Enter  4  real numbers :\n" ;
     for  (int  i  =  1 ;  i  <=  4 ;  i++) {
          cout  <<  i  <<  " : " ;
          cin  >>  a [i-1] ;
     }
     cout  <<  "Here they are in reverse order : \n" ;
     for  (i  =  3 ;  i  >=  0 ;  i--)
          cout  <<  "\ta [ "  <<  i  <<  " ]  =  "  <<  a [i]  <<  endl;
}
```

أمر التعريف double a [4] يعرف a على أنه صف من 4 عناصر كلها من النوع double . الحلقة for الأولى تسمح للمستخدم بإدخال أرقام حقيقية في هذه الأربع عناصر . بعد ذلك تقوم الحلقة for الثانية بطباعة هذه الأرقام المخزنة بترتيب عكس ترتيب إدخالهم.

وهذه عينة لتنفيذ هذا البرنامج :

```
Enter  4  real numbers :
1:   1.618
2:   2.718
3:   3.142
4:   4.444
Here they are in reverse order :
          a [3]  =  4.444
          a [2]  =  3.142
          a [1]  =  2.718
          a [0]  =  1.618
```

وسيكون الصف كالتالي :

a	1.618	2.718	3.142	4.444
	0	1	2	3

المثال التالي يعمل بنفس الطريقة ، ولكنه يستخدم ثابت رمزي لحجم الصف، وهذا يجعل تعديل البرنامج عملية سهلة .

مثال 2.5 استخدام ثابت رمزي لتعريف ومعالجة صف

```
main ()
{
     const int SIZE = 4;
     double a [SIZE];
     cout << "Enter " << SIZE << " real numbers: \n" ;
     for (int i = 1; i <= SIZE; i++) {
          cout << i << " : " ;
          cin >> a [i-1] ;
     }
     cout << "Here they are in reverse order: \n" ;
     for (i = SIZE - 1 ; i >= 0 ; i--)
          cout << "\ta [ " << i << " ] = " << a [i] << endl;
}
```

الثابت الصحيح SIZE تم اعطاؤه القيمة الابتدائية 4 . بعد ذلك تم استخدام هذا الثابت في الاعلان عن الصف a ، ومطالبة المستخدم بإدخال هذا الثابت ، وكذلك للتحكم في الحلقة for . البرنامج يعمل بنفس الطريقة كما في البرنامج السابق . الشكل العام لأمر الاعلان للصف هو :

```
type array-name [array-size];
```

حيث type هو نوع عناصر الصف ، و array-size هو عدد عناصر الصف . أمر الاعلان في مثال 2.5 كان :

```
double a [size];
```

وهذا الأمر يعرف الصف a على أنه صف من 4 عناصر من النوع double . لغة C++ القياسية تتطلب أن يكون حجم الصف array-size ثابت صحيح موجب. كما في مثال 5.2 يكون من المفيد أن نعرف حجم الصف array-size كثابت منفصل كالتالي :

```
const int size = 4 ;
```

3.5 إعطاء قيم ابتدائية للصف

في لغة ++C ، أي صف يمكن تخصيص قيماً ابتدائية له باستخدام قائمة تخصيص كالتالي :

```
float  a [4]  =  {22.2,  44.4,  66.6,  88.8} ;
```

إن القيم الموجودة في هذه القائمة يتم تخصيصها لعناصر الصف بنفس ترتيبها في القائمة .

مثال 3.5 تخصيص قيما ابتدائية لصف

هذا المثال يبين كيفية تخصيص قيماً ابتدائية لصف :

```
main  ()
{
     double  a [4]  =  {22.2 ,  44.4 ,  66.6 ,  88.8};
     for  (int  i = 0 ;  i  <  4 ;  i++)
          cout  <<  "a [ "  <<  i  <<  " ]  =  "  <<  a [i]  <<  endl;
}
```

```
a [0]  =  22.2
a [1]  =  44.4
a [2]  =  66.6
a [3]  =  88.8
```

لاحظ أن قائمة القيم الابتدائية تحتوي 4 عناصر ، وهو نفس الحجم المحدد في أمر اعلان الصف .

إذا كان الصف له عدد من العناصر أكبر من العدد الموجود في قائمة تخصيص القيم الابتدائية ، فإن العناصر المتبقية يتم وضعها أصفار.

مثال 4.5

هنا الصف له 4 عناصر ، بينما قائمة تخصيص القيم الابتدائية تحتوي عنصران فقط :

```
main ()
{
     double  a [4]  =  {22.2 ,  44.4};
     for  (int  i  = 0 ;  i  <  4 ;  i++)
          cout  <<  "a ["  <<  i  <<  " ]  =  "  <<  a [i]  <<  endl;
}
```

```
a [0]  =  22.2
a [1]  =  44.4
a [2]  =  0.0
a [3]  =  0.0
```

العنصران الأخيران في الصف اللذان ليس لهما قيماً في قائمة التخصيص، تم وضع كل منهما يساوي صفراً.

إذا كان الاعلان عن الصف لا يحتوي تخصيص له، فإن جميع عناصر الصف تأخذ قيماً غير متوقعة أو عشوائية.

مثال 5.5

في هذا المثال لم يتم تخصيص قيماً ابتدائية لعناصر الصف :

```
main ()
{
     double a [4] ;
     for (int  i  =  0;  i  <  4 ;  i++)
          cout <<  " a ["  <<  i  <<   " ] = "  <<  a [i]  << endl ;
}

a [0]  =  2.122e-314
a [1]  =  2.05154e-289
a [2]  =  3.31558e-316
a [3]  =  7.48088e-309
```

هذا يوضح أن محتويات عناصر الصف الذي لم يأخذ قيماً ابتدائية تكون غير متوقعة .

عندما يتم تخصيص قيماً ابتدائية لصف فإن الاعلان عن حجمه يمكن إهماله من أمر الاعلان ، فمثلاً في برنامج المثال 3.5 أمر الاعلان :

```
double a [4]  =  {22.2,  44.4 ,  66.6 ,  88.8} ;
```

يكافئ التعريف

```
double a [ ]  =  {22.2 ,  44.4 ,  66.6 ,  88.8} ;
```

حيث حجم الصف في هذه الحالة سيحدد بعدد القيم الموجودة في قائمة تخصيص القيم الابتدائية.

4.5 ارسال الصف إلى دالة

إن الشفرة float a [] التي تستخدم للاعلان عن صف باستخدام قائمة قيم ابتدائية تخبر المترجم شيئان : أولاً اسم الصف هو a ، وثانياً : نوع عناصر الصف سيكون float ، كما أن الرمز a يحدد عنوان

الصف في الذاكرة ، وعلى ذلك فإن الشفرة [] float a توفر جميع المعلومات التي يحتاجها برنامج المترجم لتحديد الصف. حجم الصف (أي عدد عناصره) ليس من الضروري توضيحه للمترجم .

الشفرة التي تستخدم لتمرير أو ارسال صف إلى دالة تحتوي على نوع عناصر هذا الصف واسم هذا الصف ، وهذا موضح في المثال التالي . هذا المثال يحتوي على دالتين تعالجان الصفوف. في قائمة المعاملات لكل من الدالتين . ثم تعريف الصف [] a كالتالي :

```
double  a []
```

وعدد العناصر الحقيقي سيتم امراره بواسطة متغير صحيح منفصل. عند ارسال صف لدالة بهذه الطريقة ، فإنه في الحقيقة يتم ارسال عنوان بداية الصف في الذاكرة ، وهذا العنوان يمثله اسم الصف a . بذلك تستطيع الدالة تغيير محتويات عناصر الصف بالاتصال المباشر بأماكن الذاكرة المحددة لهذه العناصر . وعلى ذلك ، فإنه بالرغم من أن اسم الصف تم ارساله كقيمة (عنوان) ، فإن عناصر هذا الصف يمكن تغيير قيمها كما لو مررت بمرجع.

مثال 6.5 دوال ادخال/اخراج الصف

هذا المثال يوضح كيفية ارسال الصفوف إلي الدوال :

```
const int SIZE  =  100 ;
void getArray (double  [] ,  int&) ;
void printArray (const double  [] ,  const int) ;
main ()
{
     double  a [SIZE] ;
     int  n;
     getArray (a , n) ;
     cout  <<  "The array has  "  <<  n  <<  "  elements : \n " ;
     printArray (a , n) ;
}
void getArray (double  a [] , int& n)
{
     n  =  0 ;
     cout  <<  "Enter data . Terminate with 0 : \n " ;
     for (n  =  0;  n  <  SIZE ;  n++)  {
          cout  <<  n  <<  " :  " ;
          cin  >>  a [n] ;
          if  (a [n]   ==  0)  break ;
     } ;
}
```

```
void printArray  (const double  a [ ] ,   const int n)
{   for (int  i  =  0 ;  i  <  n ; i++)
      cout  <<  '\t'  <<  i  <<  " : "  <<  a [i]  <<  endl ;
}
```

```
Enter data .  Terminate with 0 :
0 :  22.22
1 :  55.55
2 :  88.88
3 :  0
The array has 3 elements :
0 :  22.22
1 :  55.55
2 :  88.88
```

دالة الإدخال () getArray غيرت الشكل الرسمي للمعامل n ، وتم تمريره اليها بمرجع . المعامل a يمرر عنوان أول عنصر في الصف ، وهذا العنوان لن يتم تغييره ، ولذلك فإن a تم تمريره بقيمته ، وبما أن a هو اسم الصف (موضح بالشكل [] a) فإن الدالة ما زالت يمكنها تغيير قيم عناصر هذا الصف.

دالة الإخراج () printArray لم تعمل أي تغيير في معاملاتها ولذلك تم وصفهم في قائمة المعاملات كثوابت const .

مثال 7.5 دوال الجمع

هذه الدوال تعود بمجموع أول عدد n من عناصر أي صف

```
// Returns the sum of the first n elements of the specified array :
double sum (const double  a [] ,   const  int  n)
{  double  s = 0.0 ;
     for  (int  i  =  0;  i  <  n ;  i++)
          s  +=  a [i] ;
     return  s ;
}
```

مثل الدالة () printArray في مثال 6.5 فإن هذه الدالة لا تغير قيم معاملاتها ، ولذلك فإن كل معامل تم تمريره كثابت .

5.5 لغة ++C لا تختبر مدى الفهرس لأي صف

في بعض لغات البرمجة ، لا يسمح لمتغير الفهرس لأي صف أن تتعدى قيمته الحدود الموجودة في أمر الاعلان عن الصف . فمثلاً ، في لغة باسكال إذا تم الاعلان عن صف a حيث سيتغير فهرسه من صفر إلى 4 فإنه في هذه الحالة استخدام العنصر [5] a سيسبب توقف للبرنامج لأن 5 تقع خارج حدود فهرس هذا الصف . آلية التأمين هذه غير موجودة في ++C أو حتى C . كما يوضح المثال التالي فإن متغير فهرس الصف يمكن أن يأخذ قيماً بعيدة عن المدى المحدد بدون أن يعطي المترجم أي رسالة خطأ .

مثال 8.5 الفهرسة خارج الحدود

هنا سننفذ البرنامج السابق لجمع أول 30 عنصراً في صف مكون من 5 عناصر فقط :

```
Sum how many elements : 30
The sum of the array's first 30 elements is 8.60012e+257
```

الصف يحتوي فقط 5 عناصر ، وعندما زاد متغير الفهرس i عن القيمة 4 في الحلقة for ، فإن العنصر a[i] في هذه الحالة بدأ يتعامل مع خلايا ذاكرة ليست ضمن عناصر الصف ومحتوياتها غير متوقعة. في التنفيذ السابق قامت الدالة بجمع 5 عناصر وهو 275.75 وبعد ذلك استمرت في جمع 25 رقماً عشوائياً . الثلاثون رقماً تم جمعها لتعطي القيمة $8.10012x10^{257}$ بدون أي إشارة من الحاسب بأن هناك شيء غير صحيح .

إنها مهمة المبرمج ومسئوليته في أن يضمن عدم خروج متغير الفهرس عن حدوده . في بعض الأحوال سيخبرك الحاسب إذا خرج الفهرس عن حدوده . المثال التالي يوضح ماذا سيحدث على نظام محطة التشغيل UNIX اذا خرج الفهرس بعيداً عن حدوده .

مثال 9.5 خطأ التجزئ Segmentation Fault

في هذا التنفيذ خرج الفهرس بعيداً جداً عن حدوده بحيث أصبح خارج حدود جزء الذاكرة المحدد لتنفيذ البرنامج :

```
Sum how many elements : 300
segmentation fault
```

هذا الخطأ الحادث أثناء تنفيذ البرنامج يدل على أن النظام حاول الاتصال بجزء من الذاكرة خارج حدود الجزء المحدد لتنفيذ هذا البرنامج .

البرنامج التالي يوضح كيفية استخدام المعامل sizeof للحماية من أخطاء التعدي لمدى الفهرسة . لاستخدام هذا المعامل داخل الدالة () sum فإن الصف [ ] a لابد من الاعلان عنه كصف عالمي global .

مثال 10.5 الحماية ضد اخطاء الخروج عن مدى الفهرسة

```
double a [8] = { 5.5 , 8.8 , 2.2 , 6.6 , 9.9, 7.7 , 4.4 , 3.3}
// returns the sum of the first n elements of the array a :
double sum_a (int n)
{ if (n*sizeof (double) > sizeof (a))
        n = sizeof (a) / sizeof (double) ;
    double s = 0.0 ;
    for (int i = 0 ; i < n ; i++)
        s += a [i] ;
    return s ;
}
int main ()
{ cout << "sum_a (8) = " << sum_a (8) << endl;
    cout << "sum_a (9) = " << sum_a (9) << endl;
    return 0 ;
}
sum_a (8) = 48.4
sum_a (9) = 48.4
```

هذه الدالة تختبر حجم المعامل n . بما أن الدالة (sizeof (double تعود بحجم عناصر الصف ، فإن n ستكون خارج المدى عندما تكون n*sizeof (double) > sizeof (a) . في هذه الحالة تقوم الدالة بإعادة وضع n لتساوي عدد عناصر الصف .

6.5 خوارزيزم البحث الخطي

تستخدم الحاسبات في العادة لغرض تخزين واستعادة البيانات أكثر من أي غرض آخر ، وفي العادة تخزن البيانات في هيكل تتابعي مثل الصف . لذلك فإن أبسط طرق البحث عن هدف معين في صف تبدأ بفحص كل عنصر من عناصر هذا الصف من أوله الواحد بعد الآخر حتى يتم العثور على الهدف المطلوب . هذه الطريقة تسمى خوارزيزم البحث التتابعي .

مثال 11.5 البحث التتابعي

هذا البرنامج يختبر دالة تنفيذ خوارزم البحث التتابعي :

```
void search (int& found,  int& location ,  int a [] ,  int n ,  int target) ;

main ()
{
    int  a [] = {55, 22, 99, 66, 44, 88, 33, 77} , target , found , loc;
    do {
        cout << "Target : " ;
        cin >> target;
        search (found, loc, a , 8 , target);
        if (found) cout << target << " is at a [ " << loc << " ] . \n " ;
        else cout << target << " was not found. \n " ;
    } while (target != 0) ;
}

// Linear Search:
void search (int& found, int& location, int a [], int n, int target)
{
    found = location = 0 ;
    while (! found && location < n)
        found = (a [location++] == target) ;
    -- location ;
}
```

```
Target : 33
33 is at a [6].
Target : 44
44 is at a [4].
Target : 50
50 was not found.
Target : 0
0 was not found.
```

في كل حلقة من حلقات البحث ، فإن العنصر الحالي a [location] يتم مقارنته مع الهدف target .
تستمر الحلقة حتى يتم الحصول على الهدف ، أو نصل إلى آخر عنصر في الصف دون العثور على الهدف . كل

حلقة تزيد فهرس العنصر location بعد الاتصال به . لذلك فإنه إذا وجد الهدف المطلوب فإن الحلقة تنتهي وعندها يكون الفهرس location مساوياً فهرس العنصر المطلوب في هذه الحالة والذي تم العثور عليه.

لاحظ أن الدالة ( ) search لها ثلاث معاملات إدخال وهي a ، و n ، و target معاملان إخراج وهما found ، و location . نحن نتبع الطريقة المعتادة في إدراج معاملات الإخراج قبل معاملات الإدخال.

7.5 خوارزمية الترتيب بطريقة الفقاقيع Bubble Sorting :

خوارزم الترتيب الخطي ليس ذو كفاءة عالية ، فهو ليس الطريقة المثلى للبحث عن اسم في دليل التليفونات مثلاً. عملية البحث كعمل روتيني يمكن تنفيذها بكفاءة أفضل في الدليل لأن الأسماء تكون مرتبة ترتيباً أبجدياً في هذه الحالة . ولذلك فإنه لكي تستخدم خوارزم فعال للبحث عن معلومة في هيكل تتابعي مثل الصف، فإننا يجب أن نرتب عناصر هذا الهيكل في البداية.

هناك خوارزمات كثيرة لترتيب عناصر صف. إن خوارزم الترتيب بطريقة الفقاقيع بالرغم من أنه ليس فعال مثل خوارزمات أخرى كثيرة إلا أنه واحد من أبسط هذه الخوارزمات . يتم تنفيذ هذا الخوارزم من خلال محاولات متتابعة في كل منها يتم نقل أكبر عنصر إلى مكانه الصحيح. في كل محاولة يتم مقارنة كل عنصر بالذي يليه حيث يتم نقل الأكبر فيهم بمقدار خطوة للأمام .

مثال 12.5 الترتيب بطريقة الفقاقيع

هذا البرنامج يختبر دالة تنفذ خوارزم الترتيب بطريقة الفقاقيع. هذه الدالة تم تركيبها مع دالة الابدال swap الموضحة في مثال 15.4 :

```
void print (float  [ ] ,  const int) ;
void sort (float [ ] ,  const int) ;
main  ()
{  float  a [8]  =  {55.5,  22.5,  99.9,  66.6,  44.4,  88.8,  33.3,  77.7};
     print (a , 8) ;
     sort (a , 8) ;
     print  (a , 8) ;
}
void print (float  a [] ,  const int n)
{  for  (int  i  =  0 ;   i  <  n-1 ;  i++)
            cout  <<  a [i]  <<  " ,  " ;
     cout  <<  a [n-1]  <<  endl ;
}
```

```
void swap (float& x , float& y)
{ float temp = x ;
    x = y ;
    y = temp;
}

// Bubble sort :
void sort (float a [] , const int n)
{ for (int i = n-1 ; i > 0 ; i--)
    for (int j = 0; j < i ; j++)
        if (a [j] > a [j+1]) swap (a [j] , a [j+1])
}
```

```
55.5, 22.2, 99.9, 66.6, 44.4, 88.8, 33.3, 77.7
22.2, 33.3, 44.4, 55.5, 66.6, 77.7, 88.8, 99.9
```

تستخدم دالة الترتيب () sort حلقتين متداخلتين. الحلقة الداخلية for تقارن عنصرين متجاورين وتقوم بإبدالهم عندما يكونان في ترتيب عكسي. بهذه الطريقة فإن كل عنصر يصعد فوق (فيما يشبه الفقاعة) كل العناصر الأقل منه .

8.5 خوارزم البحث الثنائي

هذا الخوارزم يستخدم استراتيجية القسمة والأهمال حيث أنه باستمرار يقوم بقسمة الصف إلى نصفين ويهمل أحد النصفين ثم يركز عملية البحث في النصف الذي من المحتمل أن يحتوي الهدف الجاري البحث عنه .

مثال 13.5 خوارزم البحث الثنائي

هذا البرنامج يختبر دالة تنفذ خوارزم البحث الثنائي :

```
// Binary Search :
void search (int& found, int& location, int a [] , int n, int target) ;

main ()
{
    int a [] = {22, 33, 44, 55, 66, 77, 88, 99} , target , found, loc;
    do {
```

```
        cout << "Target : ";
        cin >> target ;
        search (found, loc, a, 8, target);
        if (found) cout << target << " is at a [ " << loc << " ]. \n";
        else cout << target << " was not found. \n " ;
    } while (target != 0) ;
}

void search (int& found, int& location, int a [], int n, int target)
{
    int left = 0,     right = n-1;
    found = 0;
    while (! found && left <= right) {
        location = (left + right) / 2 ;        //  the midpoint
        found = (a [location] == target) ;
        if (a [location] < target)     left = location + 1 ;
        else right = location - 1 ;
    }
}
```

```
Targer : 33
33 is at a [1].
Targer : 99
99 is at a [7].
Targer : 50
50 was not found.
Targer : 22
22 is at a [0].
Targer : 0
0 was not found.
```

في كل محاولة من محاولات الحلقة while فإن العنصر الأوسط a [location] في الصف الفرعي يحتوي العناصر من a [left] إلى a [right] يتم مقارنته مع الهدف . إذا لم يتساوى الهدف مع هذا العنصر ، فإنه إما أن يتم إهمال النصف الأيسر من الصف بوضع left = location + 1 أو يتم إهمال النصف الأيمن بوضع right = location -1 وذلك على حسب ما إذا كان a [location] < target أم لا.

إن خوارزم البحث الثنائي أكثر فعالية بكثير جداً من خوارزم البحث الخطي لأن كل محاولة تخفض حجم الصف الذي يتم البحث فيه بمقدار النصف . فمثلاً إذا كان الصف يحتوي 1000 عنصراً ، فإن البحث الخطي يحتاج إلى 1000 محاولة بينما البحث الثنائي قد لا يحتاج لأكثر من 10 محاولات لاتمام عملية البحث.

9.5 استخدام الصفوف من النوع المرقم Enumeration

الأنواع المرقمة تم وصفها في فصل 2 ، وهذه الأنواع من الطبيعي أن يتم معالجتها مع الصفوف .

مثال 14.5 ايام الاسبوع

هذا البرنامج يحدد صف اسمه high مكوناً من سبعة عناصر من النوع float تمثل درجة الحرارة العظمى للسبعة أيام في الاسبوع :

```
#include <iostream.h>
main ()
{
     enum Day { SUN, MON, TUE, WED, THU, FRI, SAT } ;
     float high [SAT+1] = {88.3, 95.0, 91.2, 89.9, 91.4, 92.5, 86.7};
     for (Day day = SUN; day <= SAT; day++)
          cout << "The high temperature for day " << day << " was "
               << high [day] << endl;
}
```

```
The high temperature for day 0 was 88.3
The high temperature for day 1 was 95.0
The high temperature for day 2 was 91.2
The high temperature for day 3 was 89.9
The high temperature for day 4 was 91.4
The high temperature for day 5 was 92.5
The high temperature for day 6 was 86.7
```

هذا البرنامج يحدد النوع Day بحيث أن أي متغير يتم الإعلان عنه على أنه من هذا النوع فسيأخذ أي قيمة من السبع قيم SUN ، MON ، TUE ، WED ، THU ، FRI ، SAT . لذلك فإن هذا النوع يمكن استخدامه بنفس طريقة استخدام النوع INT أو أي نوع آخر.

حجم الصف هو SAT+1 لأن SAT=6 والصف يحتاج لسبعة عناصر. المتغير day أعلن عنه كفهرس للحلقة for وسيأخذ القيم SUN ، MON ، ... إلخ . تذكر أنهم في الحقيقة مثل الأعداد الصحيحة 0 ، 1 ، 2 ، 3 ، 4 ، 5 ، 6 .

لاحظ أنه لا يمكن طباعة أسماء هذه الثوابت الرمزية. لذلك فإن قيم المتغير day المطبوعة بالأمر cout هي 0 ، 1 ، 2 ، ... الخ وليست SUN ، MON ، ... الخ . من مميزات استخدام الثوابت المرقمة بهذه الطزيقة أنها تجعل شفرة البرنامج تشرح نفسها . فمثلاً الحلقة for التالية :

```
for (Day  day = SUN;  day <=  SAT;  day ++)
```

تشرح نفسها كما نرى .

النوع المرقم هو في الحقيقة مثله مثل النوع short أو char ، ولكنها تختلف عنها في أنها تأخذ أسماء رمزية والقيم التي تأخذها ليس من الضروري أن تكون متتابعة . إنها في الحقيقة طريقة أخرى لاعلان قائمة من الثوابت الصحيحة. اللحق D يبين وضع الأنواع المتعددة في تسلسل الأنواع الموجودة في لغة ++C.

مثال 15.5 الأنواع البولينية Boolean

هذا المثال يبين كيفية بناء الأنواع البولينية

```
enum Boolean  { FALSE ,  TRUE  } ;

//  Prompts user for personnel information:
void getInfo (Boolean& isMarried, Boolean& spouseIsEmployed);

main ()
{
      Boolean isMarried,  spouseIsEmployed;
      getInfo ( isMarried,  spouseIsEmployed);
      if  (isMarried)  {
            cout  <<  "You are married. \n";
            if (spouseIsEmployed)  cout  <<  "Your spouse is employed. \n";
            else cout  <<  "Your spouse is not employed. \n";
      }  else cout  <<  "You are not married. \n";
}

void getInfo (Boolean& isMarried, Boolean& spouseIsEmployed)
{
      char ans ;
      cout  <<  "Are you married?  " ;  cin  >>  ans ;
```

```
        isMarried = (ans == 'Y' || ans == 'Y') ;
        if (isMarried) {
            cout << "Is your spouse employed? " ; cin >> ans ;
            spouseIsEmployed = (ans == 'Y' || ans == 'Y') ;
        } else spouseIsEmployed = FALSE;
    }
```

```
Are you married? Y
Is your spouse employed? Y
you are married.
Your spouse is employed.

Are you married? Y
Is your spouse employed? N
you are married.
Your spouse is not employed.

Are you married? N
you are not married.
```

هنا الثابت الرمزي FALSE أخذ القيمة العددية صفر ، والثابت الرمزي TRUE له القيمة العددية واحد. بهذا تصبح هذه القيم البولينية متوافقة مع لغة ++C القياسية التي تعرف القيمة صفر على أنها false (خطأ) والقيمة المختلفة عن الصفر على أنها true (حقيقية) عند استخدامها في أوامر الشروط مثل الشرط if.

10.5 تحديدات النوع

الأنواع المرقمة enumeration هي أحد الطرق المتاحة للمبرمجين لتعريف الأنواع الخاصة بهم. فمثلاً

```
enum Color { RED, ORANGE, YELLOW, GREEN, BLUE, VIOLET } ;
```

يحدد النوع color الذي يمكن استخدامه فيما بعد للإعلان عن متغيرات مثل :

```
Color shirt = BLUE;
Color car [ ] = { GREEN, RED, BLUE, RED } ;
float wavelength [VIOLET+1] = {420, 480, 530, 570, 600, 620 } ;
```

هنا shirt عبارة عن متغير يمكن لقيمه أن تأخذ أي قيمة من الست قيم المحددة في النوع color ، ولقد تم إعطاؤه القيمة الابتدائية BLUE . أما car فهو صف من 4 قيم كلها من النوع color مفهرسة من صفر إلى ثلاثة، وكذلك فإن wavelength عبارة عن صف من 6 قيم من النوع الحقيقي مفهرسة من الأحمر إلى البنفسجي.

C++ توفر أيضاً وسيلة لإعادة تسمية الأنواع الموجودة بالفعل . إن الكلمة المفتاحية typedef تعرف اسم جديد لنوع محدد، والتركيب اللغوي لذلك هو :

```
typedef type alias;
```

حيث type هو النوع المعطى ، و alias هو الاسم لهذا النوع . فمثلاً إذا كنت أحد المبرمجين المتعودين على لغة باسكال فإنك قد تحتاج لهذا التغيير :

```
typedef long Integer ;
typedef double Real ;
```

حيث بعد ذلك يمكنك استخدام الأسماء Integer و Real للإعلان عن المتغيرات من النوع long int و double كما يلي :

```
Integer  n = 22 ;
const Real PI = 3.141592653589793;
Integer frequency [64];
```

لاحظ هنا التركيب اللغوي لأمر تغيير النوع typedef لصف :

```
typedef element-type alias [] ;
```

وهذا يوضح أن عدد عناصر الصف ليست جزءاً من نوعه .

الأمر typedef لا يحدد نوعاً جديداً ، إنه فقط يعطي اسماً آخر لنوع موجود أصلاً . فمثلاً الدالة celsius المعرفة فيما سبق يمكن النداء عليها كما يلي :

```
cout << celsius (x) ;
```

حيث x تم الاعلان عنها بالأمر :

```
double x = 100;
```

ليس هناك تعارض في المعاملات لأن Real و double هي أسماء من نفس النوع ، وهذا يختلف عن الأمر enum الذي يحدد نوع جديد لعدد صحيح. المثال التالي يوضح استخدام آخر للأمر typedef .

مثال 16.5 الترتيب بالفقاقيع مرة اخرى

هذا هو نفس البرنامج الموجود في مثال 12.5 ، التغيير فقط هو في الأمر typedef على sequence الذي تم استخدامه في قائمة المعاملات وكذلك الإعلان عن a في الدالة () main :

```
typedef float sequence [ ] ;
void sort (sequence,  const int) ;
void print (const sequence,  const int);

main  ()
{
      sequence  a =  {55.5,  22.5,  99.9,  66.6,  44.4,  88.8,  33.3,  77.7};
      print  (a , 8) ;
      sort (a , 8) ;
      print (a , 8) ;
}

void swap (float& ,  float&) ;

// Bubble sort :
void sort (sequence  a ,  const  int  n)
{
      for  (int  i  = n-1 ;  i  >  0 ;  i--)
            for  (int  j  =  0 ;  j  <  i ;  j++)
                  if  (a [j]  >  a  [j+1])  swap (a [j] , a [j+1]) ;
}
void print (const sequence  a ,  const  int  n)
{
      for  (int  i  =  0;   i  <  n ;  i++)
            cout  <<  "  "  <<  a [i] ;
      cout  <<  endl;
}
```

لاحظ الأمر typedef التالي :

```
typedef float sequence [ ] ;
```

حيث القوس المربع [] يظهر بعد الاسم المرادف الجديد sequence . بعد ذلك تم استخدام الاسم الجديد بدون الأقواس المربعة للاعلان عن صفوف المتغيرات والمعاملات الرسمية.

11.5 الصفوف متعددة الأبعاد (المصفوفات)

كل الصفوف التي تعاملنا معها فيما سبق كانت كلها ذات بعد واحد. وذلك يعني أنها كلها خطية أو بمعنى آخر تتابعية . ولكن في الحقيقة فإن نوع عنصر الصف يمكن أن يكون من أي نوع تقريباً ، بما في ذلك نوع الصف نفسه. فمثلاً الصف المكون من صفوف يسمى الصف ذو الابعاد المتعددة. لذلك فإن صف ذو بعد واحد مكون من عناصر كل منها هو أيضاً صف ذو بعد واحد تمثل صف (مصفوفة) ذو بعدين . كذلك فإن الصف ذو البعد الواحد المكون من عناصر كل منها صف ذو بعدين يسمى صف ذو ثلاثة أبعاد ، وهكذا .

أبسط طريقة للإعلان عن المصفوفة هي كالتالي :

```
double  a [32]  [10] [4] ;
```

حيث هذا الأمر يعرف مصفوفة ثلاثية الأبعاد ، أبعادها هي 32 ، 10 ، 4 والأمر :

```
a [25] [8] [3]  =  99.99
```

سيخصص القيمة 99.99 للعنصر المحدد بالفهرس (25,8,3) .

مثال 17.5 قراءة وطباعة مصفوفة ذات بعدين

هذا البرنامج يبين كيفية التعامل مع مصفوفة ذات بعدين :

```
void read (int  a [] [5]) ;

void print (const  int  a [] [5]);

main ()
{
     int  a [3] [5] ;
     read  (a) ;
     print (a) ;
}

void read (int  a [] [5])
{
     cout  <<  "Enter 15 integers, 5 per row : \n" ;
     for  (int  i  =  0;  i  <  3 ;  i++) {
          cout  <<  " Row  "  <<  i  <<  "  :  " ;
          for  (int  j  =  0 ;  j  <  5 ;  j++)
               cin  >>  a [i] [j] ;
     }
}
```

```
void print (const int a [ ] [5])
{
     for (int i = 0;   i  < 3;  i++) {
          for (int j = 0;  j < 5; j++)
               cout << " " << a [i] [j] ;
          cout << endl;
}
```

```
Enter 15 integers, 5 per row :
Row 0: 44 77 33 11 44
Row 1: 60 50 30 90 70
Row 2: 85 25 45 45 55
44 77 33 11 44
60 50 30 90 70
85 25 45 45 55
```

لاحظ أنه في قائمة معاملات الدالة ، البعد الأول غير معروف من اليسار والبعد الثاني هو (5) أي معرف، وذلك لأن a مخزنة كصف ذو بعد واحد مكون من 3 صفوف لكل منها 5 عناصر ، والمترجم ليس من الضروري أن يعرف عدد هذه الصفوف الذي هو 3 ، ولكنه فقط يريد أن يعرف أن كل منها عبارة عن صف من 5 عناصر . عند إرسال مصفوفة متعددة الأبعاد إلى دالة فإن البعد الأول لا يتم تحديده وكل الأبعاد الأخرى يتم تحديدها.

مثال 18.5 قراءة وطباعة مصفوفة ذات بعدين:

```
const NUM_STUDENTS = 3 ;
const NUM_QUIZZES = 5 ;
typedef int Score [NUM_STUDENTS] [NUM_QUIZZES];
void read (Score) ;
void printQuizAverages (const Score) ;
void printClassAverages (const Score) ;

main ()
{
     Score score ;
     cout << "Enter " << NUM_QUIZZES << " score for each student : \n" ;
```

```
        read (score) ;
        cout  <<  "The quiz averages are : \n " ;
        printQuizAverages (score) ;
        cout  <<  "The class averages are : \n " ;
        printClassAverages (score) ;
}

void read (Score   score)
{
    for  (int  s  =  0 ;   s  <  NUM_STUDENT; s++)  {
        cout  <<  "student "  <<  s  <<  " : " ;
        for (int  q  =  0 ;   q  <  NUM_STUDENT ;   q++)
            cin  >>  score [s] [q] ;
    }
}

void printQuizAverages (const  Score  score)
{
    for  (int  s  =  0;   s  <  NUM_STUDENTS;  s++)   {
        float sum  =  0.0 ;
        for  (int  q  =  0 ;   q  <  NUM_QUIZZES ;  q++)
            sum   +=   score [s] [q] ;
        cout  <<  "\ tStudent  "  <<  s  <<  "  :  "   <<  sum/NUM_QUIZZES  <<  endl;
    }
}

void printClassAverages (const  Score  score)
{
    for  (int  q  =  0 ;    q  <  NUM_QUIZZES;  q++)  {
        float  sum  =  0.0 ;
        for  (int  s  =  0 ;   s  <  NUM_STUDENTS ;   s++)
            sum  +=  score [s] [q] ;
        cout  <<  " \tQuiz  "   <<  q  <<  " : "   <<  sum/NUM_STUDENTS  <<  endl ;
    }
}
```

لقد تم استخدام الأمر typedef لإعادة تسمية الاسم Score لنوع المصفوفة ذات البعدين . بذلك يصبح رأس الدالة أكثر مناسبة وأسهل في القراءة .

الدالة () printQuizAverages تطبع المتوسط لكل صف من الثلاثة صفوف الخاصة بالدرجات ، بينما الدالة () printClassAverages تطبع متوسط كل من الخمسة أعمدة الخاصة بالدرجات . هذه هي نتيجة تنفيذ هذا البرنامج :

```
Enter 5 quiz scores for each student :
Student 0 :  8  7  9  8  9
Student 1 :  9  9  9  9  8
Student 2 :  5  6  7  8  9
The quiz averages are :
            Student 0 :  8.2
            Student 1 :  8.8
            Student 0 :  7
The class averages are :
            Quiz 0 :  7.33333
            Quiz 1 :  7.33333
            Quiz 2 :  8.33333
            Quiz 3 :  8.33333
            Quiz 4 :  8.66667
```

مثال 19.5 معالجة مصفوفة ذات ثلاثة أبعاد

هذا البرنامج يعد عدد الأصفار في مصفوفة ذات ثلاث أبعاد :

```
int numZeros  (int  a [ ] [4] [3] ,  int  n1,  int  n2,  int  n3) ;
main ()
{
    int  a [2] [4] [3]  =  {  {  {5 , 0 , 2} , {0 , 0 , 9} ,  {4 , 1 , 0} , {7 , 7 , 7}  } ,
                              {  {3 , 0 , 0} , {8 , 5 , 0} , {0 , 0 , 0} , {2 , 0 , 9}  }  } ;
    cout  <<  "This array has "  <<  numZeros  (a , 2 , 4 , 3)  <<  "zeros : \n" ;
}
int numZeros  (int  a [] [4] [3] ,  int  n1 , int n2 , int n3)
{
    int count  =  0 ;
    for  (int  i  =  0;  i  <  n1;  i++)
        for  (int  j  =  0 ;  j  <  n2 ;  j++)
            for  (int  k  =  0 ;  k  <  n3 ;  k++)
                if  (a [i] [j] [k]  ==  0)  ++ count ;
    return  count ;
}
```

```
This array has 11 zeros :
```

لاحظ كيف تم إعطاء قيماً ابتدائية للمصفوفة : إنها عبارة عن صف من عنصرين ، كل منهم مكون من صف ذو 4 عناصر ، كل عنصر منها مكون من ثلاث عناصر . بذلك يكون عدد العناصر الكلي 24 عنصراً يمكن إعطاء قيماً ابتدائية لها كما يلي :

```
int   a [2] [4] [3]  =  {5, 0, 2, 0, 0, 9, 4, 1, 0, 7, 7, 7, 3, 0, 0, 8, 5, 0, 0, 0, 0, 2, 0, 9} ;
```

أو كما يلي :

```
int   a [2] [4] [3] = {  {5, 0, 2, 0, 0, 9, 4, 1, 0, 7, 7, 7} , { 3, 0, 0, 8, 5, 0, 0, 0, 0, 2, 0, 9}  } ;
```

ولكن كل من هاتين الطريقتين صعب القراءة وصعب الفهم عن طريقة إعطاء القيم الابتدائية لقائمة ثلاثية الأبعاد.

لاحظ أيضاً الثلاث حلقات for المتداخلة . عامة فإن مصفوفة ذات عدد من الأبعاد d تعالج بعدد من d من الحلقات المتداخلة حيث تخصص حلقة لكل بعد .

أسئلة للمراجعة

1.5 كم عدد الأنواع المختلفة التي يمكن أن تأخذها عناصر أي صف ؟

2.5 ما هو نوع ومدى الفهرس لأي صف ؟

3.5 ما هي القيم التي ستأخذها عناصر صف إذا تم الاعلان عنه ولكن لم يتم اعطاء قيماً ابتدائية له ؟

4.5 ما هي القيم التي ستأخذها عناصر صف إذا تم الاعلان عنه ولكن تم اعطاء قيماً ابتدائية لعدد أقل من العناصر المحدد للصف ؟

5.5 ماذا سيحدث اذا كان عدد العناصر في أمر اعطاء القيم الابتدائية أكبر من حجم الصف ؟

6.5 كيف يختلف الأمر enum عن الأمر typedef ؟

7.5 عند إرسال صف ذو أبعاد متعددة إلي دالة، لماذا تطلب C++ أن تحدد كل الابعاد إلا البعد الأول في قائمة المعاملات ؟

مسائل برمجة محلولة

8.5 اكتب ونفذ برنامج يقرأ عدد غير محدد من الارقام، وبعد ذلك اطبع هذه الارقام مع بعد كل منها عن المتوسط. يمكننا أن نجمع هذه الأعداد أثناء قراءتها وبعد ذلك نحسب المتوسط لها بقسمة هذا المجموع على عددهم :

```
const int SIZE = 100;
main ( )
{ double a [SIZE], x, sum = 0.0;
     int n;
     cout << "Enter data.    Terminate with 0 : \n" ;
     for (n = 0;  ; n ++) {
          cin >> x ;
          if (x == 0)  break ;
          a [n] = x ;
          sum += x ;
     } ;
double mean = sum/n ;
cout << "mean = "  << mean << endl ;
for (int i = 0 ; i < n ; i++)
     cout << '\t' << a [i] << '\t' << a [i] - mean << endl ;
}
```

```
Enter data. Terminate with 0 :
1.23
7.65
0
mean = 4.44
1.23    -3.21
7.65     3.21
```

تستمر حلقة الادخال حتى قراءة صفر ، والبعد عن المتوسط يتم طباعته بعد حسابه كالتالي :

a [i] - mean

5.9 أكتب واختبر الدالة

void insert (int a [ ] , int& n , int x)

هذه الدالة تقوم بإدخال العنصر x في الصف المرتب a المكون من عدد n من العناصر ثم تزيد n بمقدار واحد . العنصر الجديد يتم إدخاله في المكان الذي يحافظ على الصف مرتباً ، وهذا يتطلب إزاحة للعناصر للأمام لتدبير مكان للعنصر الجديد x . البرنامج الذي سنختبر به هذه الدالة يحدد صف من 100 عنصر تم إعطاء قيماً ابتدائية لعشرة منها في ترتيب تصاعدي :

```
void print (int [ ] ,  int) ;
void insert (int [ ] , int& , int) ;
```

```
main ()
{
    int a [100] = { 261, 288, 289, 301, 329, 333, 345, 346 ,346, 350 } ;
    int n = 10, x ;
    print (a , n) ;
    cout << " Item to be inserted : " ;
    cin >> x ;
    insert (a, n, x) ;
    print (a , n) ;
}

void print (int a [ ] , int n)
{
    for (int i = 0 ; i < n-1 ; i++) {
        cout << a [i] << " , " ;
        if ((i+1) %16 == 0) cout << endl ;
    }
    cout << a [n-1] << endl ;
}

void insert (int a [ ] , int& n , int x)
{
    for (int i = n ; i > 0 && a [i-1] > x ; i--)
        a [i] = a [i-1] ;
    a [i] = x ;
    ++n ;
}
```

```
261, 288, 289, 301, 329, 333, 345, 346, 346, 350
Item to be inserted : 300
261, 288, 289, 300, 301, 329, 333, 345, 346, 346, 350

261, 288, 289, 301, 329, 333, 345, 346, 346, 350
Item to be inserted : 400
261, 288, 289, 301, 329, 333, 345, 346, 346, 350, 400

261, 288, 289, 301, 329, 333, 345, 346, 346, 350
Item to be inserted : 200
200, 261, 288, 289, 301, 329, 333, 345, 346, 346, 350
```

الدالة ( ) insert تعمل من النهاية العليا للصف ، وتبحث في الاتجاه العكسي عن المكان الصحيح لوضع العنصر x . في أثناء البحث تقوم بإزاحة العناصر الأكبر من x مكان واحد ناحية اليمين لإخلاء مكان للعنصر x . في التنفيذ الأول العدد 300 ثم وضعه في المكان المناسب بعد إزاحة 7 عناصر ناحية اليمين.

التنفيذ الثاني والثالث تختبر قيماً على الحدود أو على أطراف الصف. أحد هذه الحدود هو عندما يكون العنصر المراد إدخاله أصغر من كل عناصر الصف، وهذا تم اختباره بإدخال الرقم 200.

10.5 اكتب واختبرالدالة

```
int frequency  (float  a [] ,  int  n  ,   int  x)
```

هذه الدالة تعد عدد مرات ظهور العنصر x في أول عدد n من عناصر الصف a وتعود بهذا العدد على أنه تردد أو تكرار العنصر x في a . هنا تم بدأ صف a من 40 رقماً صحيحاً مرتبة عشوائياً لاختبار هذه الدالة :

```
int frequency (float  [ ] ,  int  ,  int) ;

main  ( )
{
     float  a [ ]  =  { 561,  508, 400,  301,  329, 599, 455, 400, 346,  346,  329,
                        375, 561, 390, 399,  400,  401,  561,  405,  405,  455, 508,
                        473, 329, 561,  505,  329,  455,  561,  599,  561,  455,
                        346,  301,  455,  561,  399,  599,  508,  508} ;
     int  n  =  40,  x;
     cout  <<  " Item:  " ;
     cin  >>  x ;
     cout  <<  "The frequency of item  "  <<  x  <<  "  is  "
          <<  frequency  (a,  n,  x)  <<  endl;
}

int frequency  (float  a [ ] ,  int  n ,  int  x)
{
     int count  =  0 ;
     for  (int  i  =  0 ;  i  <  n ;  i++)
          if  (a [i]  ==  x)  ++count ;
     return  count ;
}
```

```
Item :  508
The frequency of item  508  is  4

Item :  500
The frequency of item  500  is  0
```

تستخدم الدالة العداد count ، وتقارن كل عنصر من عناصر الصف مع العنصر x وتزيد العداد بمقدار واحد في كل مرة يتساوى فيها العنصران .

11.5 كون الخوارزم Insertion sort ، حيث في هذا الخوارزم تسير الحلقة الأساسية من 1 إلى n-1 وفي المحاولة رقم i يتم إدخال العنصر a [i] في مكانه الصحيح في الصف الجزئي a [0] إلى a [i] . يتم ذلك بإزاحة كل عناصر الصف الجزئي التي تكون أكبر من العنصر a [i] بمقدار مكان واحد ناحية اليمين. بعد ذلك يتم نسخ العنصر a [i] في المكان الواقع بين العنصر a [i] والأماكن الأكبر منه . (انظر المسألة 9.5) . سنختبر هذه الدالة عن طريق بدأ صف a من 8 أرقام مرتبة عشوائياً :

```
void print (float  [ ] ,   const  int)  ;
void sort  (float  [ ] ,  const  int)  ;

main  ( )
{
    float  a [8]  =  {88.8,  44.4,  77.7,  11.1,  33.3,  99.9,  66.6,  22.2} ;
    print  (a ,  8) ;
    sort  (a ,  8) ;
    print  (a ,  8) ;
}

void print  (float  a [ ] ,   const  int  n)
{
    for  (int  i  =  0 ;   i  <  n-1 ;  i++)  {
        cout  <<  a [i]  <<  " ,  " ;
        if  ( (i+1) % 16  ==  0)   cout  <<  endl ;
    }
    cout  <<  a [n-1]  <<  endl;
}

/ /  Insertion Sort :
void sort  (float  a [ ] ,  const  int  n)
```

```
{
    float temp ;
    for (int  i = 1 ;  i  <  n ;  i++ )    { //  sort  {a [0] , ... ,  a [i] } :
        temp  =  a [i] ;
        for (int  j = i ;  j  > 0  && a [j-1] >  temp ;  j--)
            a [j]  = a [j-1] ;
        a [j]  =  temp ;
    }
}
```

```
88.8,  44.4,  77.7,  11.1,  33.3,  99.9,  66.6,  22.2
11.1,  22.2,  33.3,  44.4,  66.6,  77.7,  88.8,  99.9
```

في المحاولة رقم i من الحلقة الأساسية في الخوارزم Insertion sort سيتم ادخال العنصر a [i] بحيث يكون الصف الجزئي { a [0] , ... , a [i] } مرتباً ترتيباً تصاعدياً . يتم ذلك بتخزين العنصر a [i] مؤقتاً في المتغير temp وبعد ذلك تستخدم الحلقة الداخلية لازاحة العناصر الأكبر لليمين باستخدام a [j] = a [j-1] ، وبعد ذلك يوضع المتغير temp في العنصر a [j] . لاحظ أن a [k] ≤ a [j] لكل k ≤ j ، a [j] ≤ a [k] لكل j ≤ k ≤ i ، بهذا نضمن أن الصف الجزئي { a [0] , ... , a [i]} يصبح مرتباً. عند الانتهاء من المحاولة الأخيرة في الحلقة الأساسية i == n-1 يكون عندها الصف { a [0] , ... , a [n-1] } مرتباً.

12.5 أعد كتابة واختبر الدالة Bubble sort المقدمة في مثال 12.5 كترتيب غير مباشر بدلاً من تحريك عناصر الصف الحقيقية ، رتب صف الفهرس بدلاً من ذلك .

برنامج اختبار هذه الدالة يفترض صف a يبدأ ببعض الأرقام العشوائية ، وصف فهرس index يبدأ بالعناصر index [i] = i بذلك نضمن أن العنصر a [index [i]] سيكون هو العنصر a [i] في البداية:

```
void print (const float a [ ] ,  const  int n) ;
void sort (float  a [ ] ,  int index [ ] ,  int  n) ;
void print (const  float  a [] ,  int   index [ ] ,  const  int  n) ;

main  ( )
{
    float  a [8]  =  {55, 22, 99,  66, 44, 88,  33,  77} ;
    int  index  [8]  =  { 0,  1,  2,  3,  4,  5,  6,  7 } ;
    print  (a,  8) ;
    sort  (a , index,  8) ;
    print  (a ,  index,  8) ;
    print  (a,  8) ;
```

```
}

void swap (int& , int&) ;

// Indirect Bubble Sort :
void sort (float a [] ,  int index [] , int n)
{
    for (int i = 1 ;  i  < n ; i++)
        for (int j = 0 ;  j  < n-1 ;  j++)
            if (a [index [j] ] > a [index [j+1] ] )
                swap (index [j] , index [j+1]) ;
}

void print (const float a [] ,  const  int n)
{
    for (int  i = 1 ;  i  <  n ; i++)
        cout <<  "  "  << a [i] ;
    cout  <<  endl ;
}
void print (const float a [] , int index  [] , const int n)
{
    for (int i = 0 ;  i  < n ;  i++)
        cout << "  "  <  a [index [i] ] ;
    cout << endl ;
}
```

```
55  22  99  66  44  88  33  77
22  33  44  55  66  77  88  99
55  22  99  66  44  88  33  77
```

التعديل الوحيد الذي نحتاجه للدالة Bubble sort هو احتواء كل فهرس مع [...] index . بمعنى أن الفهرس j تم استبداله بالعنصر index [j] ، والفهرس j+1 تم استبداله بالعنصر index [j+1] . تأثير ذلك هو أن الصف سيترك كما هو دون تغيير وبدلاً من ذلك سنحرك عناصر الصف index .

لاحظ أنه لدينا دالتي طباعة () print زائدتي التحميل overloaded ، أحدهما لطباعة الصف مباشرة والأخرى لطباعته بطريقة غير مباشرة باستخدام صف الفهرس . بذلك نتأكد من أن الصف الأصلي a ترك كما هو ولم يتغير نتيجة عملية الترتيب غير المباشر .

13.5 ابني الدالة Sieve of Eratosthenes (المنخل) لإيجاد الأعداد الأولية . جهز صف prime [n] من الأعداد الصحيحة واجعل a [0] = a [1] = 0 (صفر وواحد ليست أعداداً أولية). واجعل العناصر

a [2] حتى a [n-1] تساوي 1 . بعد ذلك لكل قيمة i من 3 حتى n-1 إجعل a [i] = 0 إذا كانت i تقبل القسمة على 2. (بمعنى أن i% 2 = 0) . بعد ذلك لكل قيمة i من 4 حتى n-1 اجعل a [i] = 0 . اذا كانت i تقبل القسمة على 3 . كرر ذلك لجميع الأرقام المقسوم عليها من 2 حتى n/2 . بعد الانتهاء من ذلك فإن كل القيم i التي مازال العنصر المقابل لها a [i] يساوي 1 تكون هي الأعداد الأولية ، هذه الأرقام تعتبر هي الأرقام التي سقطت من المنخل. برنامج الاختبار يفترض صف اسمه prime من الف عنصر كلها أصفار، وبعد النداء على الدالة sieve () فإنها ستطبع الفهارس i التي لها prime [i] == 1 :

```
const int SIZE = 500 ;
void sieve (int prime [ ] , const int n) ;

main ( )
{
    int prime [SIZE] = {0} ;
    sieve (prime , SIZE) ;
    for (int i = 0 ;  i < SIZE ;  i++) {
        if (prime [i] )  cout << i <<  " " ;
        if ( (i+1) % 50  == 0)  cout << endl;
    }
    cout << endl ;
}

//  Sets prime [i] = 1 if and only if i is prime :
void sieve (int prime [] , const int n)
{
    for (int i = 2 ;  i < n ; i++)
        prime [i] = 1 ;        // assume all i > 1 are prime
    for (int p = 2 ;  p <= n/2 ;  p++) {
        for (int m = 2*p;  m < n;  m += p)
            prime [m] = 0 ;    // no multiple of p is prime
        while (!prime [p])
            ++p;     // advance p to next prime
    }
}
```

```
2 3 5 7 11 13 17 19 23 25 29 31 35 37 41 43 47 49
53 55 59 61 65 67 71 73 77 79 83 85 89 91 95 97
101 103 107 109 113 115 119 121 125 127 131 133 137 139 143 145 149
151 155 157 161 163 167 169 173 175 179 181 185 187 191 193 197 199
203 205 209 211 215 217 221 223 227 229 233 235 239 241 245 247
251 253 257 259 263 265 269 271 275 277 281 283 287 289 293 295 299
301 305 307 311 313 317 319 323 325 329 331 335 337 341 343 347 349
353 355 359 361 365 367 371 373 377 379 383 385 389 391 395 397
401 403 407 409 413 415 419 421 425 427 431 433 437 439 443 445 449
451 455 457 461 463 467 469 473 475 479 481 485 487 491 493 497 499
```

الدالة sieve () في البداية تضع prime [i] = 1 لكل $i \geq 2$ ، وبعد ذلك تعيد وضع prime [i] = 0 مرة أخرى لكل تكرار m من العدد الأولي p .

14.5 اكتب واختبر الدالة :

```
void reverse (float a [] , int n)
```

هذه الدالة تعكس عناصر صف بحيث يصبح آخر عنصر فيه هو الأول والعنصر الثاني يصبح العنصر قبل الأخير ، وهكذا . لاحظ أن ذلك يختلف عن مثال 1.5 الذي لا يتطلب تحريك أي عنصر من عناصر الصف.

هذا الحل يستبدل كل عنصر في النصف الأول من الصف (n/2) مع ما يقابلها من النصف الثاني من الصف :

```
void print (const float [ ] , const int) ;
void reverse (float [ ] , const int) ;

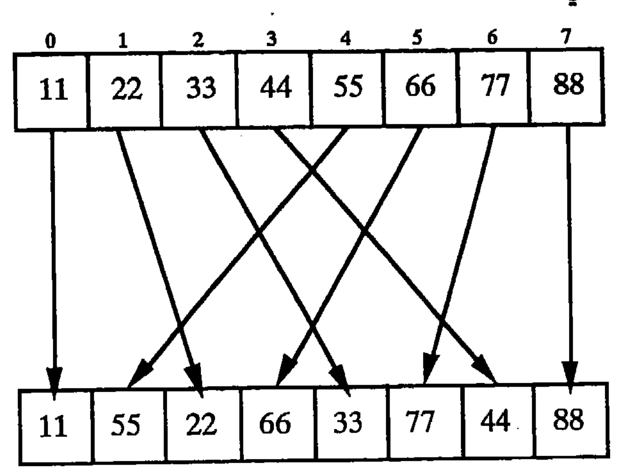
main ( )
{
    float a [8] = { 88.8, 44.4, 77.7, 11.1, 33.3, 99.9, 66.6, 22.2} ;
    print (a , 8) ;
    reverse (a , 8) ;
    print (a , 8) ;
}

void reverse (float a [ ] , const int n)
{
    float temp ;
```

```
    for (int i = 0;  i < n/2; i++) {
        temp = a [i] ;
        a [i] = a [n-i-1] ;
        a [n-i-1] = temp ;
    }
}
```

88.8, 44.4, 77.7, 11.1, 33.3, 99.9, 66.6, 22.2
22.2, 66.6, 99.9, 33.3, 11.1, 77.7, 44.4, 88.8

15.5 اكتب واختبر دالة تقوم بتفنيط عناصر صف مكون من عدد زوجي من العناصر . كمثال على ذلك تقوم الدالة باستبدال الصف {11, 22, 33, 44, 55, 66,77,88} بالصف {11, 55, 22, 66, 33, 77, 44, 88} كما يلي :



تقوم هذه الدالة بإدخال عناصر النصف الثاني من الصف في عناصر النصف الأول منه. من السهل عمل ذلك باستخدام صف مؤقت يسمى temp وبعد ذلك يتم نسخ هذا الصف المؤقت في الصف a :

```
/ / The perfect shuffle for an even number of elements :
void shuffle (float a [] , conts int n)
{   float temp [n] ;
    for (int i = 0 ;  i < n/2 ;  i++)
    {   temp [ 2*i] = a [i] ;
        temp [2*i+1] = a [n/2+i] ;
    }
    for (i = 0; i < n ; i++)
        a [i] = temp [i] ;
}
```

فـي حالـة 8 == n فإن الحـلـقة for الأولى تنـسخ [o] a فـي [0] temp و [4] a في [1] temp عندما 0 == i ، وبعد ذلك تنسخ [1] a في [2] temp و [5] a في [3] temp عندما 1 == i ، بعد ذلك تنسخ [2] a فـي [4] temp و [6] a فـي [5] temp عندما 2 == i ، بعد ذلك تنسخ [3] a في [6] temp و [7] a في [7] temp عندما 3 == i .

16.5 اكتب واختبر الدالة التي تقوم بدوران عناصر مصفوفة ثنائية الأبعاد مكونه من أرقام بمقدار 90 درجة في اتجاه عقارب الساعة كمثال على ذلك ستقوم الدالة بتحويل المصفوفة .

```
11  22  33
44  55  66
77  88  99
```

إلى المصفوفة

```
77  44  11
88  55  22
99  66  33
```

هذا الحل يفترض أن النوع matrix تم تحديده بالأمر typedef .

```
void rotate (Matrix m , const int n)
{
    Matrix temp ;
    for (int i = 0 ; i < SIZE ; i++)
        for (int j = 0; j < SIZE ; j++)
            temp [i] [j] = m [SIZE - j - 1] [i] ;
    for (i = 0 ; i < SIZE ; i++)
        for (j = 0 ; j < SIZE ; j++)
            m [i] [j] = temp [i] [j] ;
}
```

سنستخدم مصفوفة مؤقتة لتخزين عناصر المصفوفة التي تم دورانها، وبعد ذلك ننسخها مرة ثانية في المصفوفة m. في حالة 3 == n تقوم الحلقة for الأولى بنسخ [0] [2] m في [0] [0] temp ، m [1][0] في [1] [0] temp ، [0] [0] m في [2] [0] temp وذلك عندما 0 == i ، وبعد ذلك ننسخ m [1] [2] في [0] [1] temp ، و [1] [1] m في [1] [1] temp ، و [1] [0] temp في [2] [1] temp عندما 1 == i، وبعد ذلك ننسخ [2] [2] m في [0] [2] temp ، و [2] [1] m في [1] [2] temp ، و [2] [0] m في [2] [2] temp عندما 2 == i.

مسائل برمجة اضافية

17.5 اكتب واختبر برنامج مثل المثال 2.5 ولكن في هذه الحالة يملأ الصف بالعكس وبعد ذلك يطبعه بالترتيب الذي خزن به . فمثلاً أول عنصر يتم قراءته يخزن في آخر مكان ويكون الأخير في الطباعة .

18.5 اكتب واختبر برنامج مثل المسألة 8.5 ولكن في هذه الحالة يحسب ويطبع كل من المتوسط والبعد عن المتوسط للبيانات المدخلة . إن البعد عن المتوسط لعدد n من الأعداد a n-1 , ... ,a0 يحدد بالعلاقة :

$$\sigma = \frac{\sqrt{\sum_{i=0}^{n-1}(a_i - \mu)^2}}{n-1}$$

حيث μ هي المتوسط . هذه العلاقة تعني أن نربع كل عنصر (a [i] - mean) ، ثم نجمع هذه المربعات، ثم نأخذ الجذر التربيعي لهذا المجموع ثم نقسم هذا الجذر على (n-1) .

19.5 طـــور برنامج المســـألة 18.5 بحيث يحسب ويطبع الـ Z-scores للبيانات المدخلة. الـ Z-scores للبيانات a0 حتى a n-1 تعطي بالعلاقة التالية :

$$Z_i = (a_i - \mu) / \sigma$$

إن الـ Z-scores تعمم normalize البيانات بحيث تصبح مركزة حول الصفر وتبعد عن المتوسط بمقدار واحد.

20.5 في الأيام الماضية كان التقدير C هو التقدير المتوسط، وكان المدرسون في الفصول الكبيرة يحسبون منحنيات التقديرات على حسب التوزيع التالي :

A: $1.5 \leq z$

B: $0.5 \leq z < 1.5$

C: $-0.5 \leq z < 0.5$

D: $-1.5 \leq z < -0.5$

F: $z < -1.5$

إذا كانت التقديرات لها شكل الجرس (normall distribution) ، فإن الخوارزم سينتج في هذه الحالة 7 ٪ تقدير A ، 24 ٪ تقدير B ، 38 ٪ تقدير C ، 24 ٪ تقدير D ، و 7 ٪ تقدير F . هنا قيم z تمثل الـ Z-scores الموصوفة في المسألة 19.5 . طور البرنامج في المسـألة 18.5 بحيث يطبع منحنيات التقدير للدرجات المدخلة.

21.5 اكتب واختبر دالة تستبدل كل الأرقام السالبة في صف مكون من أرقام صحيحة بقيمهم المطلقة :

22.5 اكتب واختبر دالة تعود بالقيمة الصغرى المخزنة في صف .

23.5 اكتب واختبر دالة تعود برقم المكان (الفهرس) الذي يحتوي القيمة الصغرى في صف.

24.5 اكتب واختبر الدالة التالية التي تعود بعنوان كل من القيمة العظمى والقيمة الصغرى في صف .

```
void extremes (int& min , int& max , int a [] , int n)
```

25.5 اكتب واختبر الدالة التالية التي تعود بعنوان كل من القيمة العظمى والقيمة التالية لها في صف (من المحتمل أن تكونا متساويين) .

```
void largest (int& max1 , int& max2, int a [] , int n)
```

26.5 اكتب واختبر الدالة التالية التي تحاول أن تحذف عنصر من عناصر صف :

```
int remove (int a [] , int& n , int x)
```

الدالة تبحث في أول عدد n من عناصر الصف a عن العنصر x . إذا وجد العنصر x ، يتم حذفه ، وكل العناصر فوق هذا العنصر يتم ازاحتها لأسفل، وينقص n بمقدار واحد، وتعود الدالة بالقيمة 1 للدلالة على أنه تم ازاحة العنصر . إذا لم يوجد العنصر x فإن الصف يترك كما هو ، وتعود الدالة بالقيمة صفر للدلالة على أن العنصر غير موجود في الصف (انظر المسألة 9.5) .

27.5 اكتب واختبر الدالة التالية :

```
void rotate (int a [] , int n , int k)
```

هذه الدالة تقوم بدوران أول عدد n من عناصر الصف a بمقدار k من الأماكن ناحية اليمين (أو k- من الأماكن ناحية اليسار إذا كانت k سالبة) . آخر عدد k من العناصر يتم دورانهم إلى بداية الصف. كمثال على ذلك إذا كان a هو الصف :

0	1	2	3	4	5	6	7
22	33	44	55	66	77	88	99

فإنه بعد النداء rotate (a, 8, 3) سيحول الصف a إلى

0	1	2	3	4	5	6	7
77	88	99	22	33	44	55	66

لاحظ أن النداء rotate (a, 8, -5) سيكون له نفس التأثير .

28.5 اكتب واختبر الدالة التالية :

```
void append (int a [] , int m , int b [] , int n)
```

هذه الدالة تلحق أول عدد n من عناصر الصف b في نهاية أول عدد m من عناصر الصف a . تفترض الدالة أن الصف a به أماكن لعدد m + n من العناصر على الأقل . فمثلاً إذا كان كل من a و b كما يلي:

a	0	1	2	3	4	5	6	7	8	9	10	11	12	13
	22	27	33	34	39	44	50	55	0	0	0	0	0	0

b	0	1	2	3	4	5	6	7
	66	72	77	88	90	0	0	0

فإن النداء (append (a, 8, b, 5 سيجعل الصف a كما يلي :

a	0	1	2	3	4	5	6	7	8	9	10	11	12	13
	22	27	33	34	39	44	50	55	66	72	77	88	90	0

لاحظ أن الصف b يترك بدون تغيير ، وخمس عناصر فقط من a هي التي تغيرت .

29.5 اكتب واختبر الدالة التالية :

```
int is palindrame (int a [] , int n)
```

هــذه الدالــة تعـود بواحد أو صفر على حسب ما إذا كان أول عدد n من العناصر في الصف a تكون palindrame بالبندروم ، حيث الباليندروم هو مصفوفة تقرأ من اليمين كما تقرأ من الشمال مثل الصف {22, 33, 44, 55, 44, 33, 22} . تحذير :

هذه الدالة يجب أن تترك الصف بدون تغيير .

30.5 اكتب واختبر دالة تجمع العناصر المتناظرة في صفين كل منهما ذو بعد واحد ومكون من أعداد صحيحة ولهما نفس عدد العناصر . فمثلاً إذا كان الصفين هما :

22 33 44 55

7 4 1 -2

فإن الصف الناتج يكون

29 37 45 53

31.5 اكتب واختبر دالة تطرح صفين كل منهما ذو بعد واحد ومكون من عناصر صحيحة والصفين لهما نفس الحجم . فمثلاً إذا كان الصفين كما يلي :

22 33 44 55

7 4 1 -2

فإن الصف الثالث الناتج سيكون

15 29 43 57

32.5 اكتب واختبر دالة تضرب صفين كل مهما ذو بعد واحد ومكون من أرقام صحيحة والصفين لهما نفس الحجم، فمثلاً إذا كان الصفين كما يلي :

2 4 6 8

7 4 1 -2

فإن الصف الثالث الناتج يكون

14 16 6 -16

33.5 أحد الأسباب التي تجعل صورة خوارزم الترتيب بالفقاقيع الموجودة في مثال 12.5 غير فعال هو أنه يحتاج لنفس العدد من المقارنات للصف المكون من n من العناصر مهما كانت درجة ترتيب هذا الصف قبل بدأ الخوارزم . حتى أنه إذا كان الصف مرتباً تماماً في البداية فإن هذا الخوارزم سيأخذ حوالي $n^2/2$ عدد من المقارنات. عدل هذا الخوارزم بحيث أن الحلقة الرئيسية while تتوقف بمجرد أن تتم دورة كاملة دون الحاجة إلى عملية ابدال للعناصر. استخدم علم flag . (بمعنى متغير صحيح يوقف الحلقة عندما تكون قيمته واحد) يسمي sorted وهذا العلم تكون قيمته صفراً في بداية كل دورة من الحلقة الأساسية، ثم تغير قيمته إلى واحد بمجرد أن تتم أي عملية إبدال للعناصر .

34.5 أعد كتابة واختبر الدالة () sort الموجودة في مثال 12.5 مستخدماً الترتيب بالاختيار selection sort بدلاً من الترتيب بالفقاقيع . يعتمد خوارزم الترتيب بالاختيار لصف مكون من n من العناصر على مسح هذه العناصر n-1 من المرات، وفي كل مرة يبحث عن العنصر الكبير التالي a [j] ويضعه في المكان المخصص له . ففي الدورة الأولى يختار اكبر العناصر ويستبدله مع العنصر a [n-2] (الأخير) ، وفي الدورة الثانية يختار ثاني أكبر عنصر من العناصر a [0] حتى a [n-2] ويستبدله مع العنصر a [n-2] ، وهكذا في المحاولة الأخيرة يبحث عن أكبر عنصر من العناصر المتبقية a [0] حتى a [n-1] ويستبدله مع العنصر a [n-i] .

35.5 ابني دالة الترتيب بالاختيار الغير مباشرة (انظر المسألة 12.5) .

36.5 ابني دالة الادخال الغير مباشرة (انظر المسألة 11.5) .

37.5 اكتب واختبر دالة تحسب القيمة الوسطى median في صف مرتب . القيمة الوسطى هي الرقم الأوسط .

38.5 اكتب واختبر دالة تحسب القيمة k ٪ من صف مرتب . القيمة k ٪ في صف هي الرقم الذي تبعد قيمته k ٪ من بداية الصف . فمثلاً القيمة 75 ٪ هي القيمة التي يوجد قبلها 75 ٪ من الأرقام التي تكون قيمتها أقل من هذا الرقم . القيمة الوسطى median هي القيمة التي يوجد قبلها 50 ٪ من الأرقام التي قيمتها أقل من القيمة الوسطى.

39.5 اكتب برنامج يحسب عدد مرات التفنيط التام لعناصر صف تلزم حتى يعود هذا الصف إلى ترتيبه الأصلي (انظر المسألة 15.5) .

40.5 اكتب برنامج التفنيط التام لصف مكون من أي عدد من الأرقام الزوجية أو الفردية.

41.5 اكتب واختبر الدالة التالية :

```
void prepend (int a [] , int m, int b [], int n)
```

هذه الدالة تضع أول عدد n من عناصر الصف b أمام أول عدد m من عناصر الصف a . الدالة تفترض أن الصف a به أماكن كافية لعدد m + n من العناصر على الأقل .

42.5 اكتب واختبر الدالة التي تقلب الصفوف أعمدة والأعمدة صفوف transpose في مصفوفة ذات بعدين مربعة مكونة من عناصر صحيحة . فمثلاً تقوم هذه الدالة بتحويل المصفوفة

```
11  22  33
44  55  66
77  88  99
```

إلى المصفوفة

```
11  44  77
22  55  88
33  66  99
```

43.5 اكتب واختبر الدالة التي تصفر عناصر قطري مصفوفة مربعة ذات بعدين مكونة من عناصر صحيحة . كمثال على ذلك تقوم الدالة بتحويل المصفوفة :

```
11  12  13  14  15
21  22  23  24  25
31  32  33  34  35
41  42  43  44  45
51  52  53  54  55
```

إلى المصفوفة

0	12	13	14	0
21	0	23	0	25
31	32	0	34	35
41	0	43	0	45
0	52	53	54	0

44.5 اكتب واختبر الدالة التي تعود بأثر trace أو (مجموع عناصر القطر الرئيسي لمصفوفة) مربعة ذات بعدين من الأرقام الصحيحة . فمثلاً هذه الدالة ستعود بالقيمة 46 في حالة المصفوفة :

11	22	33
40	20	60
35	25	15

45.5 اكتب واختبر دالة تقارن عناصر مصفوفتين كل منهما ذات بعدين ولهما نفس العدد من العناصر. هذه الدالة تكون مصفوفة ثالثة كل عنصر فيها يكون -1 أو صفر أو 1 على حسب ما إذا كان العنصر المقابل في المصفوفة الأولى أكبر من أو يساوي أو أقل من العنصر المقابل في المصفوفة الثانية . فمثلاً نفرض أن لدينا المصفوفتان التاليتان :

22	44	66		33	44	55
50	50	50		50	50	80

فإن المصفوفة الناتجة ستكون

-1	0	1
0	0	-1

46.5 اكتب واختبر دالة تحسب الضرب الخارجي outer product لصفين من العناصر الصحيحة. العنصر (i, j) للمصفوفة ذات البعدين الناتجة يكون نتيجة ضرب العنصر i في الصف الأول في العنصر j في الصف الثاني . فمثلاً لو أن لدينا الصفين التاليين

20 30 40 و 3 -2

فإن المصفوفة الناتجة ستكون

60	90	120
-40	-60	-80

47.5 القيمة الصغرى / العظمى minimax في مصفوفة ذات بعدين هي العنصر الأصغر في صفه والأكبر في عموده. القيمة العظمى /الصغرى maxmin هي العكس من ذلك : أي العنصر الذي يكون

الأكبر في صفه والأصغر في عموده . القيم العظمى / الصغرى والصغرى / العظمى تسمى نقط البردعة . فمثلاً في المصفوفة

33	11	22	44	44
55	99	55	66	77
66	77	33	88	22

يكون العنصر 44 == [3] [0] a هو نقطة عظمى/صغرى لأنها أكبر قيمة في الصف رقم صفر وأصغر قيمة في العمود رقم 3 . كذلك العنصر 55 == [2] [1] a هو قيمة صغرى/عظمى لأنه أصغر عنصر في الصف رقم 1 ، وهو أيضاً أكبر عنصر في العمود رقم 2 . أكتب واختبر برنامج يقرأ الرقمين n ، m ويعد ذلك يقرأ المصفوفة ذات البعدين n × m ، ثم بعد ذلك يطبع أماكن نقاط البردعة في هذه المصفوفة. فمثلاً البرنامج يطبع الخرج التالي :

a [0 , 2] = 44 is a maxmin

a [1 , 1] = 55 is a minmax

وذلك في حالة المصفوفة السابقة .

48.5 اكتب واختبر دالة تولد مثلث باسكال لمصفوفة يتم ارسالها لهذه الدالة . فمثلاً إذا ارسلنا المصفوفة a والرقم 4 لهذه الدالة فإنها ستعود بما يلي :

1	0	0	0	0
1	1	0	0	0
1	2	1	0	0
1	3	3	1	0
1	4	6	4	1

الاجابة على أسئلة المراجعة

1.5 نوع واحد فقط : أي أن جميع عناصر الصف لابد أن تكون من نفس النوع .

2.5 فهرس الصف لابد أن يكون من النوع الصحيح وفي المدى من صفر حتى n-1 حيث n هي عدد عناصر الصف .

3.5 في حالة عدم اعطاء قيم ابتدائية ، فإن عناصر الصف ستأخذ قيماً عشوائية غير متوقعة.

4.5 اذا كان عدد العناصر التي ستأخذ قيماً ابتدائية أقل من حجم الصف، فإن هذه القيم الابتدائية ستخصص لأول عدد من عناصر الصف وأما العدد الباقي من عناصر الصف فإنها تلقائياً ستأخذ القيمة صفر .

5.5 انه من الخطأ أن تعطي عدد من القيم الابتدائية أكبر من حجم الصف .

6.5 الأمر enum يحدد نوع عددي جديد يكون نوع صحيح ليس له اشارة . الأمر typedef يحدد مرادف لنوع موجود أصلاً.

7.5 عند ارسال مصفوفة ذات أكثر من بعد إلى دالة ، فإن كل الأبعاد ماعدا البعد الأول لابد من تحديدها حتى يستطيع المترجم حساب مكان كل عنصر في المصفوفة.

الفصل السادس

6

المؤشرات والمراجع

Pointer and References

1.6 مقدمة :

عند الإعلان عن أحد المتغيرات ترتبط به ثلاثة قرائن أساسية : اسمه ، نوعه ، وعنوانه بالذاكرة. كمثال الإعلان

```
int n ;
```

يربط الاسم n ، والنوع int ، والعنوان بالذاكرة الذي يخزن به قيمة n لنفترض أن العنوان 0x3fffd14 (في النظام الستعشري والموضح بالملحق G) عندها يمكن رؤية n كالتالي

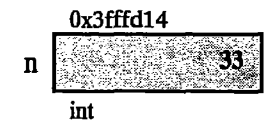
0x3fffd14

n

int

يمثل الصندوق مكان تخزين المتغير بالذاكرة . اسم المتغير على اليسار ، عنوان المتغير من أعلى ، ونوع المتغير أسفل الصندوق.

إذا كانت قيمة المتغير معروفة فتكون موضحة بداخل الصندوق :



يمكن التعامل مع قيمة المتغير بواسطة اسمه . فمثلاً يمكن طباعة قيمة المتغير n بالأمر التالي:

```
cout << n ;
```

عنوان المتغير يمكن التعامل معه بعامل العنوان & address operator . فمثلاً يمكن طباعة عنوان المتغير n بالأمر التالي :

```
cout << &n;
```

عامل العنوان & "يعمل" على اسم المتغير لينتج العنوان . ان لها اسبقية تنفيذ 15 (انظر الملحق C) حيث أن له نفس اسبقية عامل النفي المنطقي Not وعامل الزيادة المسبقة ++ .

مثال 1.6 طبع قيم المؤشر

يبين هذا المثال كيف نطبع كلا من القيمة والعنوان لمتغير

```
main ()
{
    int  n = 33 ;
    cout << "n = " << n << endl ;    //  print the value of n
    cout << "&n = " << &n << endl ;  //  print the address of n
}
```

ويكون خرج البرنامج كالتالي

```
n = 33
&n = 0x3fffd14
```

يمكنك القول أن السطر الثاني للخرج 0x3fffd14 هو عنوان بالعلامة البادئة "ox" للصورة الستعشرية. هذا العنوان يساوي العدد العشري 67,108,116 (انظر الملحق G) اظهار العنوان للمتغير بهذه الطريقة ليست له فائدة كبيرة . عامل العنوان له استعمالات أخرى أكثر أهمية . لقد رأينا استخدام واحد له في الفصل الرابع: يرمز أو يشير للثوابت المرجعية في اعلان الدالة هذا الاستخدام يقترب كثير من استخدام آخر وهو اعلان المتغيرات المرجعية.

2.6 المراجع References :

المرجع هو اسم مرادف لمتغير آخر . يعلن عنها باستخدام العامل المرجعي & والذي يلحق بنوع المرجع.

مثال 2.6 استخدام المراجع

هنا يعلن عن r على أنها مرجع إلى n :

```
main ()
{
    int n = 33 ;
    int& r = n;   //  r is a reference for n
    cout << "n = " << n << ", r = " << r << endl;
    --n ;
    cout << "n = " << n << ", r = " << r << endl;
    r *= 2 ;
    cout << "n = " << n << ", r = " << r << endl;
}
```

```
n = 33,    r = 33
n = 32,    r = 32
n = 64,    r = 64
```

الميزان n ، r هما اسمان مختلفان لنفس المتغير. دائماً لهما نفس القيمة . انقاص قيمة n يغير قيمة كل من n ، r إلى 32 . مضاعفة r يزيد كلاً من n ، r إلى 64 .

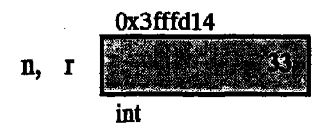
مثال 3.6 المراجع هي مرادفات

يبين هذا المثال أن r ، n لهما نفس العنوان بالذاكرة :

```
main()
    {
        int  n = 33;
        int& r = n;
        cout << "&n =  " << &n << ",  &r = "  << &r << endl;
    }
```

```
&n = 0x3fffd14,    &r = 0x3fffd14
```

التخطيط التالي يوضح كيف تعمل المراجع



تخزن القيمة 33 مرة واحد فقط . الميزين n ، r هما اسماء رمزية لنفس المكان بالذاكرة 0x3fffd14 مثل الثابت ' const ' لابد من تحديد المرجع عند اعلانه . هذا المطلب يبدو معقولاً : المرادف لابد من شيء ينسب إليه. بمعنى أن المرجع لابد له من شيء يرجعْ اليه.

معاملات المرجع عرفت للدوال (الفصل الرابع) . نرى الآن أنها تعمل بنفس الطريقة كالمتغيرات المرجعية: إنها مجرد اسماء بديلة لمتغيرات أخرى . بالتأكيد العامل المرجعي لدالة في الحقيقة هو متغير مرجعي نطاقه محدد بالدالة.

لقد رأينا أن العلامة & لها استخدامات عديدة في C++ : عندما تتقدم اسم المتغير فإنها تعيد عنوان المتغير . عند استعمالها بعد النوع في اعلان المتغير فإنها تعلن هذا المتغير على أنه مرادف للمتغير الذي حددت قيمته ، وعند استعمالها بعد النوع في اعلان معاملات الدالة فإنها تعلن المعامل ليكون معامل مرجعي للمتغير

المرسل لها . كل هذه الاستخدامات تمثل اختلافات لنفس النمط: إن علامة & تشير إلى العنوان الذي تخزن فيه القيمة.

3.6 المؤشرات Pointers

العامل المرجعي " & " يعيد عنوان المتغير في الذاكرة . لقد استعملنا هذه الخاصية في المثال 1.6 لطباعة العنوان. يمكن أيضاً تخزين العنوان في متغير آخر. نوع المتغير الذي يخزن العنوان يسمى المؤشر. إذا كان المتغير من النوع int فإن متغير المؤشر لابد أن يكون له النوع مؤشر لرقم صحيح "pointer to int" . ويرمز له بـ ; *int

مثال 4.6 قيم المؤشرات هي عناوين

```
main ()
{
    int n = 33 ;
    int* p = &n;   // p holds the address of n
    cout << "n = " << n << ", &n = " << &n << ", p = " << p << endl ;
    cout << "&p = : << &p << endl;
}
```

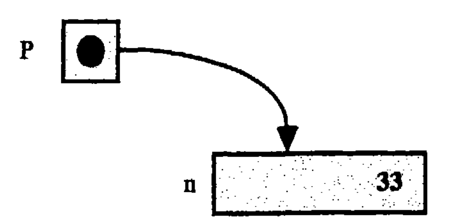
```
n = 33,  &n = 0x3fffd14,  p = 0x3fffd14
&p = 0x3fffd10
```

متغير المؤشر p والتعبير &n لهما نفس النوع (مؤشر إلى .int) ونفس القيمة (0x3fffd14). هذه القيمة مخزنة في المكان 0x3fffd10 بالذاكرة.



يطلق على المتغير p "مؤشر" لأن قيمته "تشير" إلى موضع قيمة أخرى. أنه من النوع int لأن القيمة التي يشير إليها هي .int .

قيمة المؤشر هي عنوان . هذا العنوان يعتمد على حالة الماكينة (الحاسب) الذي يجري عليه البرنامج . في معظم الحالات تكون القيمة الفعلية لهذا العنوان (مثل 0x3fffd14) غير مهمة للمبرمج ولذلك فإن التخطيط السابق عادة ما يتم رسمه كالآتي :



وذلك يتضمن الخصـائص الأساسية لـ p و n : p هو المؤشر لـ n و n لها القيمة 33 . يمكن التفكير في المؤشر على أنه "محدد وضع" حيث يبين أين توجد قيمة أخرى.

غالباً سنحتاج إلى استعمال المؤشر p وحده للحصول على القيمة التي يشير إليها ويطلق على ذلك "إعادة المرجعية" للمؤشر. وتتم ببساطة عن طريق تطبيق النجمة * كمؤشر على المؤشر.

مثال 5.6 إعادة المرجعية للمؤشر Dereferencing a Pointer

تشير p هنا إلى الرقم الصحيح المسمى n ، ولذا p* و n لهما نفس القيمة

```
main ()
{
    int n = 33;
    int* p = &n;  // p points to n
    cout << "*p = " << *p << endl;
}
```

```
*p = 33
```

وهذا يبين ان p* هي مرادف لـ n .

عامل العنوان & وعامل اعادة المرجعية * هما عكس بعضهما p* == n عندما p == &n. ويمكن التعبير أيضاً عن ذلك بـ n* == &n ايضاً p == &*p .

مثال 6.6 "المرجعية" هي عكس "اعادة المرجعية"

هنا p تشير إلى الرقم الصحيح المسمى n و r هو مرجع تحددت قيمته بالمكان الذي تشير اليه p . ولهذا p مرجع لـ n و r اعادة مرجعية p . اذن r هي مرادف لـ n . بمعنى أنهما اسمان مختلفان لنفس القيمة 33.

```
main ()
{
    int n = 33;
    int* p = &n;  // p points to n
    int& r = *p;  // r is a reference for n
    cout << "r = " << r << endl;
}
```

```
r = 33
```

وذلك يوضح أن r هي مرجع لـ n

4.6 الانواع المشتقة Derived Types

في المثال 6.6 p كان لها النوع مؤشر إلى .int بينما r لها النوع مرجع إلى .int هذه الأنواع هي مشتقة من النوع .int مثل المصفوفات والثوابت والدوال هذه أنواع مشتقة . فيما يلي بعض الاعلانات للأنواع المشتقة:

```
int& r = n;                    // r has type reference to int
int* p = &n;                   // p has type pointer to int
int a [] = {33, 66};           // a has type array of int
const int c = 33;              // c has type const int
int f () = { return 33; } ;    // f has type function returns int
```

أنواع ++C تصنف إما اساسية أو مشتقة . (انظر الملحق D). تشتمل الأنواع الأساسية على أنواع الترقيم وكافة أنواع الأرقام وأما كل نوع مشتق فيتوقف على نوع آخر . المتغير الذي أعلنت عنه ليأخذ أي نوع من الأنواع المشتقة الموضحة سابقاً (ثابت، مصفوفة، مؤشر، مرجع أو دالة) يرتكز على نوع اساسي واحد . النوع المشتق الذي يرتكز على أكثر من نوع اساسي واحد يسمى نوع هيكلي . تلك التي تشتمل على هياكل ، اتحادات أو طبقات كما سندرس في الفصول التالية.

5.6 الأهداف والقيم اليسارية

بيعتا المرجع (Ellis) للغة ++C يعرف "الهدف بأنه منطقة تخزين والقيمة اليسارية هي تعبير يشير إلى هدف أو دالة." أساساً المصطلحات lvalue و rvalue تنسب إلى اشياء ظهرت على يسار أو يمين التنسيب ولكن القيمة اليسارية lvalue هي الآن أكثر عامية.

انهل .إن أبسط أمثلة على هذه المصطلحات هي اسماء أهداف (أي متغيرات) :

```
int n ;
n = 44 ;    // n is an lvalue
```

() main

وأبسط الأمثلة على الاشياء التي ليست قيم يسارية هي الحروف

```
44 = n;   //ERROR :   44 is not an lavlue
```

ولكن الثوابت الرمزية هي قيم يسارية

```
const int MAX = 65535 ;  // Max is an lvalue
```

{

رغم عدم ظهورها على يسار التنسيب

```
MAX = 21024 ;      // ERROR : MAX is a constant
```

القيم اليسارية التي تظهر على الجانب الأيسر من التحديد تسمى القيم اليسارية الغير قابلة للتخفيف mutable lvalues . المتغير يكون ذو قيمة يسارية مخففة، بينما يكون الثابت غير قابل للتخفيف. أمثلة أخرى على القيمة اليسارية المخففة تتضمن المتغيرات الرقمية والمؤشرات المعاد المرجعية اليها.

```
int  a [8] ;
a [5]  =  22 ;            //  a [5]  is a mutable lvalue
int  *p  =  &n ;
*p  =  77 ;               //  *p is a mutable lvalue
```

أمثلة أخرى للقيم اليسارية الغير مخففة تتضمن المصفوفات ، الدوال والمراجع. بصفة عامة القيمة اليسارية هي أي شيء يمكن الحصول على عنوانه. حيث أن العنوان هو ما يحتاجه المتغير المرجعي عند اعلانه ، فإن تعبير ++C يتطلب عند اعلانه تحديد القيمة اليسارية :

```
type&  refname  =  lvalue ;
```

كمثال ، هذه هي الطريقة الصحيحة لاعلان مرجع

```
int&  r  =  n ;       //  OK: n is an lvalue
```

ولكن الطرق التالية غير صحيحة :

```
int&  r  =  44;            //  ERROR:  44 is not an lvalue
int&  r  =  n++;           // ERROR:  n++ is not an lvalue
int&  r  =  cube  (n) ;  // ERROR : cube  (n)  is not an lvalue
```

6.6 إعادة المرجع

نوع العائد من الدالة يمكن أن يكون مرجع بشرط أن تكون القيمة العائدة قيمة يسارية وعلى ألا تكون محلية للدالة. هذا التحديد يعني أن القيمة المعادة هي في الحقيقة مرجع للقيمة اليسارية التي توجد بعد انتهاء الدالة. وبالتالي فإن هذه القيمة اليسارية العائدة يمكن استعمالها كأي قيمة يسارية أخرى وكمثال على الجانب الأيسر من التنسيب :

مثال 7.6 اعادة المرجع

```
int& max (int&  m,   int&  n)       // return type is reference to int
{
     return  (m  >  n ?   m : n) ;   //  m and n are non-local references
}
```

```
main ()
{
    int m = 44,   n = 22;
    cout << m << " , "  << n << " , " << max (m, n) << endl;
    max (m, n) = 55 ;       // changes the value of m from 44 to 55
    cout << m << " , "  << n << " , " << max (m, n) << endl;
}
```

```
44,   22,   44
55,   22,   55
```

الدالة max () تعيد مرجع للأكبر من المتغيرين المرسلين للدالة. حيث أن القيمة المعادة هي مرجع فالتعبير max (m, n) يعمل كمرجع بالنسبة لـ (m) (حيث m أكبر من n) . لهذا فإن تخصيص 55 للتعبير max (m, n) يكافئ تخصيصها لـ m ذاتها .

مثال 8.6 استعمال الدالة كدليل Subscript للمصفوفة

```
float& component (float* v, int k)
{
    return v [k-1] ;
}
main ()
{
    float v [4] ;
    for (int k = 1 ;  k <= 4; k++)
        component (v, k) = 1.0/k ;
    for (int i = 0 ; i < 4 ; i++)
        cout << "v ["  << i << "] = "  << v [i] << endl;
}
```

ويكون شكل الخرج على النحو التالي :

```
v [0] = 1
v [1] = 0.5
v [2] = 0.333333
v [3] = 0.25
```

تسمح الدالة component () بالوصول للمتجهات باستخدام التعبير العلمي " 1 – اساس الترقيم" بدلاً عن التلقائي "صفر– اساس الترقيم" لذلك فالتنسيب component (v, k) = 1.0/k هو في الحقيقة التنسيب v [k +1] = 1.0/k وسنرى طريقة أفضل لعمل ذلك في الفصل التاسع.

7.6 المصفوفات والمؤشرات

رغم أن أنوا ع المؤشرات ليست أعداد صحيحة إلا أن بعض العمليات الحسابية للأعداد الصحيحة يمكن تطبيقها على المؤشرات . تأثير هذه الحسابات هو جعل المؤشر يشير إلى مكان آخر في الذاكرة . التغيير الحقيقي في العنوان يعتمد على حجم النوع الأساسي الذي يشير إليه المؤشر.

المؤشرات يمكن زيادتها وانقاصها مثل الأعداد الصحيحة ومع ذلك فإن الزيادة والنقص في قيمة المؤشر يساوي حجم الهدف الذي يشير إليه.

مثال 9.6 التنقل في المصفوفة باستخدام المؤشر

يبين هذا المثال كيف يستخدم المؤشر للتجول داخل المصفوفة

```
main ()
{
    const int SIZE = 3 ;
    short a [SIZE] = {22, 33, 44} ;
    cout << "a = " << a << endl ;
    cout << "sizeof (short) = " << sizeof (short) << endl ;
    short* end = a + SIZE;    // converts SIZE to offset 6
    short sum = 0 ;
    for (short* p = a; p < end; p++) {
        sum += *p;
        cout << "\t p = " << p ;
        cout << "\t *p = " << *p ;
        cout << "\t sum = " << sum << endl ;
    }
    cout << "end = " << end << endl ;
}
```

```
a = 0x3fffd1a
sizeof (short) = 2
        p = 0x3fffd1a        *p = 22        sum = 22
        p = 0x3fffd1c        *p = 33        sum = 55
        p = 0x3fffd1e        *p = 44        sum = 99
end = 0x3fffd20
```

في السطر الثاني للخرج ترى أن الأرقام من النوع short في هذه الماكينة تحتل 2 بايت . وحيث أن p هو مؤشر ذو النوع short ، فكل مرة يزاد تتقدم 2 بايت إلى الرقم الصحيح short التالي في المصفوفة بهذه الطريقة

*p += sum يتراكم مجموع الأرقام الصحيحة . إذا كانت p مؤشر إلى النوع double وكان size of (double) هو 8 بايت فإن كل مرة تزاد فيها p : فإنها تتقدم بمقدار 8 بايتات.

لمثال 9.6 يبين أنه عندما يزاد مؤشر فإن قيمته تزيد بمقدار الحجم size (بالبايت) في الهدف الذي يشير اليه. وعلى سبيل المثال :

```
float  a [8] ;
float*  p  =  a ;                 // p points to a [0]
++p;                              // increases the value of p by sizeof (float)
```

إذا كانت floats تحتل 4 بايت فإن ++p تزيد قيمة p بمقدار 4 كما أن ;5 =+ p تزيد قيمة p بمقدار 20 . هذه هي الكيفية التي نتجول بها في المصفوفة : وذلك يبدأ المؤشر عند أول قيمة بالمصفوفة ثم نزيد المؤشر بصورة متتالية. كل زيادة تحرك المؤشر إلي العنصر التالي في المصفوفة.

يمكن أيضاً استخدام مؤشر للوصول مباشرة داخل المصفوفة . كمثال على ذلك ، يمكن الوصول إلى العنصر [5] a يبدأ المؤشر عند [0] a ثم نضيف القيمة 5 إليها

```
float*  p  =  a    // p points to a [0]
p  +=  5 ;         // now  p  points to a [5]
```

بمجرد بدأ المؤشر لعنوان البداية في المصفوفة فإنه يعمل كدليل.

تحذير : في ++C يمكن الوصول وحتى تعديل محتويات مكان بالذاكرة لم يسبق تخصيصه. إن ذلك مخاطرة ويجب تجنبها بصورة عامة. على سبيل المثال

```
float  a [8] ;
float*  p  =  a [7] ;       // p points to last element in the array
++p;                        // now p points to memory past last element!
*p  =  22.2;                //  TROUBLE!
```

المثال التالي يوضح علاقة أقوى بين المصفوفات والمؤشرات : اسم المصفوفة في حد ذاته هو const مؤشر لأول عنصر في المصفوفة. كما أنه يبين أنه يجوز مقارنة المؤشرات .

مثال 10.6 فحص عناوين عناصر المصفوفة

```
main ()
{
    short  a [ ]  =  {22,  33,  44,  55,  66} ;
    cout  <<  "a  =  "  <<  a  <<  ",  *a  =  "  <<  *a  <<  endl ;
    for  (short*  p  =  a;   p  <  a  +  5 ;  p++)
        cout  <<  "p  =  "  <<  p  <<  ",  *p  =  "  <<  *p  <<  endl ;
}
```

```
a = 0x3fffd08,   *a = 22
p = 0x3fffd08,   *p = 22
p = 0x3fffd0a,   *p = 33
p = 0x3fffd0c,   *p = 44
p = 0x3fffd0e,   *p = 55
p = 0x3fffd10,   *p = 66
```

بدايةً و a و p متشابهان : كلاهما مؤشرات لـ short ولهما نفس القيمة (0x3fffd08). حيث أن a هو مؤشر ثابت فلا يمكن زيادته ليتجول في المصفوفة. بدلاً عن ذلك نزيد p ونستعمل شرط الخروج p < a+5 لإنهاء الحلقة for . هذا يحسب a+5 على أنها العنوان الستعشري

0x3fffd08 + 5 * size of (short) = 0x3fffd08 + 5 * 2 = 0x3fffd08 +0xa = ox3fffd12

وتستمر الحلقة طالما قيمة p أقل من 0x3fffd12

عامل فهرس المصفوفة [ ] يكافئ عامل اعادة المرجعية * . إنه يمكن من الوصول المباشر داخل المصفوفة بنفس الطريقة :

```
a [0] == *a
a [1] == * (a+1)
a [2] == * (a+2)
```

وهكذا

ولذا يمكن التجول داخل المصفوفة كالاتي

```
for (int i = 0; i < 8; i++)
    cout << * (a+i) << endl;
```

المثال التالي يوضح كيف تستخدم المؤشرات مع الأعداد الصحيحة للتحرك للأمام والخلف داخل الذاكرة.

مثال 11.6 مضاهاة النموذج

في هذا المثال تفحص الدالة loc خلال n1 من العناصر الأولى في المصفوفة a1 بحثاً عن سلسلة أعداد صحيحة مخزنة n2 من العناصر الأولى في المصفوفة a2 بداخلها. إن وجدت . فإنها تعيد مؤشر إلى مكان في a1 حيث يبدأ a2 . وإلا فإنها تعيد مؤشر NULL (المؤشر الصفري) .

```
short* loc (short* a1, short* a2, int n1, int n2)
{
     short* endl = a1 + n1 ;
     for (short* p1 = a1 ; p1 < endl ; p1++)
          if (*p1 == *a2)    {
               for (int j = 0 ;  j < n2 ; j++)
                    if (p1 [j]  != a2 [j]) break ;
               if (j == n2) return p1 ;
          }
     return 0;
}

main ()
{
     short a1 [9] = {11, 11, 11, 11, 11, 22, 33, 44, 55} ;
     short a2 [5] = {11, 11, 11, 22, 33}
     cout << "Array a1 begins at location\t " << a1 << endl;
     cout << "Array a2 begins at location\t " << a2 << endl;
     short* p = loc (a1, a2, 9, 5) ;
     if (p) {
          cout << "Array a2 found at location\t " << p << endl;
          for (int i = 0;  i < 5 ; i++)
               cout << "\t" <<   &p [i]  <<   " : "  <<   p [i]
                    <<  "\t" <<   &a2 [i]  <<   " : "  <<  a2 [i] << endl ;
     }
     else cout  <<  "Not found. \n"  ;
}
```

```
Array  a1 begins at location          0x3fffd12
Array  a2 begins at location          0x3fffd08
Array  a2 found at location           0x3fffd16
               0x3fffd16 : 11     0x3fffd08 : 11
               0x3fffd18 : 11     0x3fffd0a : 11
               0x3fffd1a : 11     0x3fffd0c : 11
               0x3fffd1c : 22     0x3fffd0e : 22
               0x3fffd1e : 33     0x3fffd10 : 33
```

خوارزم مضاهاة النموذج يستخدم حلقتين. الحلقة الخارجية محكومة بواسطة المؤشر p1 الذي يشير إلى أعضاء المصفوفة a1 . أما الحلقة الداخلية فتبدأ في اختبار التشابه مع المصفوفة a2. الحلقة الداخلية محكومة بواسطة الرقم الصحيح j الذي يستخدم في المقارنة بين عنصرين متناظرين في المصفوفتين . إذا وجد عدم

تطابق ، تنتهي الحلقة الداخلية وتستمر الحلقة الخارجية بزيادة p1 للبحث عن تطابق في العضو التالي من a1 .
إذا سمح للحلقة الداخلية أن تنتهي فإن الشرط (j == n2) سيكون صحيح ويؤخذ المكان الحالي الذي يشير إليه p1.

في الاختبار نتبين أن التوافق قد وجد فعلاً عن طريق اختبار العناوين الفعلية .

8.6 العامل new :

عند اعلان مؤشر مثل :

```
float*  p ;    //  p  is a pointer to float
```

فإنها فقط تخصص ذاكرة للمؤشر نفسه. قيمة المؤشر ستكون عنوان بالذاكرة ولكن الذاكرة في هذه اللحظة لم يتم تخصيصها . هذا يعني أن التخزين قد يكون مستخدماً بواسطة متغير آخر. في هذه الحالة لم يتم بدأ قيمة لـ p ، أي لا تشير إلى ذاكرة تم تخصيصها. من الخطأ محاولة الوصول إلى الذاكرة التي تشير إليها.

```
*p  =  3.14159;       // ERROR:  no storage has been allocated for  *p
```

لتجنب هذه المشكلة يجب بدأ المؤشرات عند اعلانها :

```
float  x  =  3.14159;   //  x contains the value  3.14159
float*  p  =  &x;       //  p contains the address of x
cout  <<  *p ;          //  OK:  *p has been allocated
```

في هذه الحالة لا توجد مشكلة للوصول إلى p* حيث أن الذاكرة المطلوبة لتخزين float 3.14159 تخصصت تلقائياً عند اعلان x وأن p تشير إلى نفس الذاكرة المخصصة .

طريقة أخرى لتجنب جنوح المؤشر هو تخصيص صريح لذاكرة المؤشر نفسه. يتم ذلك باستخدام العامل :new

```
float*  q ;
q  =  new float;    // allocates storage for 1 float
*q  =  3.14159;     //  OK:  *q has been allocated
```

العامل new يعيد عنوان لمجموعة (Block) من s بايتات غير مخصصة بالذاكرة . حيث s هو حجم لـ (float).

(عادة size of (float) يكون 4 بايت) . تنسيب هذا العنوان إلى q يضمن أن q* غير مستخدم حالياً بواسطة أي متغير آخر.

السطرين الأولين يمكن ضمهما بتحديد بدأ q عند اعلانها .

```
float *q = new float ;
```

لاحظ أن استخدام العامل new لبدأ q يبدأ المؤشر نفسه فقط وليس الذاكرة التي يشير إليها. يمكن تنفيذ المهمتين في نفس الخبر الذي يعلن عن المؤشر.

```
float* q = new float (3.14159) ;
cout << *q ;          // ok: both q and *q have been initialized
```

في هذه الحالة الغير متوقعة التي لا توجد ذاكرة كافية لتخصيص قطاع (Block) بالحجم المطلوب فإن العامل new يعيد "0" (المؤشر الصفري NULL) :

```
double* p = new double ;
if (p == 0) abort () ;  // allocator failed: insufficient memory
else *p = 3.141592658979324 ;
```

هذه الشفرة المحكمة تنادي الدالة abort () لتمنع اعادة المرجعية للمؤشر الصفري NULL.

اعتبر ثانية البديلين لتخصيص الذاكرة :

```
float x = 3.14159 ;                  // allocates named memory
float* p = new float (3.14159) ;     // allocates unnamed memory
```

في الحالة الأولى تخصص الذاكرة وقت الترجمة للمتغير المسمى x . في الحالة الثانية تخصص الذاكرة وقت التنفيذ إلى هدف غير مسمى نصل إليه من خلال *p .

9.6 عامل الحذف delete

العامل delete يعكس مفعول العامل new ، حيث يحرر الذاكرة المخصصة . تطبق فقط على المؤشرات المخصصة صراحة بالعامل new :

```
float* q = new float (3.14159) ;
delete q ;                     //deallocates q
*q = 2.71828                   // ERROR : q has been deallocated
```

الغاء تخصيص q يعيد حيز من البايتات sizeof (float) إلى الذاكرة الحرة ، ويجعلها ميسرة للتخصيص للأهداف الأخرى. بمجرد تحرير q يجب الا تستخدم ثانية الا بعد اعادة تخصيصها. المؤشر المحرر ايضاً يسمى dangling pointer مثل المؤشر الذي لم تحدد بدايته . لا يشير إلى أي شيء.

المؤشر لثابت لا يمكن حذفه

```
const int *p = new int;
delete p ;           // ERROR: cannot delete pointer to const
```

يتفق هذا القيد مع المبدأ العام أنه لا يمكن تغيير الثوابت

استخدام عامل الحذف delete للأنواع الاساسية (الخ char, int, float, double) لا ينصح به عامة حيث أن المكسب قليل في مقابل مخاطرة بالوقوع في خطأ جسيم .

```
floatfloat x = 3.14159 ;        //x contains the value 3.14159
float* p = &x;                  // p contains the address of x
delete p;                       // RISKY: p was not allocated by new
```

إن هذا سيحرر المتغير x ، وهذا خطأ يمكن أن يكون بالغ الصعوبة في الاكتشاف.

10.6 المصفوفات الديناميكية

اسم المصفوفة هو مؤشر ثابت يخصص له ذاكرة في وقت الترجمة :

```
float a [20] ;                 // a is a const pointer to a block of 20 floats
float* const p = new float [20] ;          // so is p
```

هنا كلاً من a ، p هما مؤشرات ثابتة لحيز مكون من 20 عدد حقيقي (float) اعلان a يسمى ربط استاتيكي static binding لأنها مخصصة في وقت الترجمة.

يقتصر الرمز على الذاكرة المخصصة حتى ولو لم تستخدم المصفوفة اثناء تنفيذ البرنامج.

على النقيض يمكننا استخدام مؤشر غير ثابت لتأجيل تخصيص الذاكرة حتى وقت تنفيذ البرنامج. عادة ما تسمى ربط وقت التنفيذ أو الربط الديناميكي :

```
float *p = new float [20] ;
```

المصفوفة المعلن عنها بهذه الطريقة تسمى مصفوفة ديناميكية .

قارن بين الطريقتين لتعريف المصفوفة

```
float a [20] ;                  // static array
float *p = new float [20] ;  // dynamic array
```

المصفوفة الاستاتيكية a تكونت في وقت الترجمة ؛ تبقى الذاكرة الخاصة بها خلال تنفيذ البرنامج. المصفوفة الديناميكية p تكونت في وقت تنفيذ البرنامج ؛ تخصص لها الذاكرة فقط عند تنفيذ اعلانها، اكثر من ذلك فإن الذاكرة المخصصة للمصفوفة p تحرر بمجرد استدعاء عامل الحذف delete عليها .

```
delete [ ] p ;  // deallocates the array p
```

لاحظ ان عامل الفهرسة [ ] لابد أن يوجد بهذه الطريقة حيث أن p هي مصفوفة .

مثال 12.6 استخدام المصفوفة الديناميكية

الدالة () get هنا توجد مصفوفة ديناميكية .

```
void get (double*& a,  int& n)
{
    cout << "Enter number of items: " ;  cin >> n ;
    a = new double [n] ;
    cout << "Enter " << n << " items, one per line : \n " ;
    for (int i = 0 ; i < n ; i++) {
        cout << "\t" << i+1 << " : " ;
        cin >> a [i] ;
    }
}

void print (double* a, int n)
{
    for (int i = 0 ; i < n ; i++)
        cout << a [i] << "  " ;
    cout << endl ;
}

main ()
{
    double* a;           // a is simply an unallocated pointer
    int n ;
    get (a, n) ;         // now a is an array of n doubles
    print (a, n) ;
    delete [ ] a ;       // now a is simply an unallocated pointer again
    get (a, n) ;         // now a is an array of n doubles
    print (a, n) ;
}
```

```
Enter number of items: 4
Enter 4 items, one per line:
        1 : 44.4
        2 : 77.7
        3 : 22.2
        4 : 88.8
44.4  77.7  22.2  88.8
Enter number of items: 2
Enter 2 items, one per line:
        1 : 3.33
        2 : 9.99
3.33  9.99
```

بداخل الدالة get () ، يخصص العامل new أماكن تخزين لـ n double بعد الحصول على قيمة n تفاعلياً . لذا فإن المصفوفة تكونت "فوريا" اثناء تنفيذ البرنامج. من قبل استخدمت get () لتكوين مصفوفة أخرى لـ a . المصفوفة الحالية يجب تحريرها باستخدام العامل delete . لاحظ أن عامل الفهرسة [ ] يجب أن يستخدم عند حذف المصفوفة.

لاحظ أن معامل المصفوفة a هو مؤشر تم ارساله بالمرجع :

```
void get (double*&a ,  int&  n)
```

إن ذلك ضرورياً حيث أن العامل new سيغير قيمة a التي هي عنوان العنصر الأول للمصفوفة المخصصة حديثاً.

11.6 استخدام const مع المؤشرات

المؤشر لثابت يختلف عن المؤشر الثابت. هذا الاختلاف موضح في المثال التالي :

مثال 13.6 : مؤشرات ثابتة ومؤشرات لثوابت ومؤشرات ثابتة لثوابت

هذا المقطع يعلن عن أربع متغيرات : مؤشر p ، مؤشر ثابت cp ، مؤشر لثابت pc ومؤشر ثابت cpc لثابت:

```
int  n  =  44 ;                  // an int
int*  p  =  &n ;                 // a pointer to an int
++ (*p) ;                        // ok:  increments int  *p
++ p ;                           //  ok:  increments pointer  p
int*  const  cp  =  &n;          //   a  const pointer to an int
++ (*cp) ;                       //  ok: increments int *cp
++ cp;                           //  illegal: pointer cp is const
const  int  k  =  88 ;           // a const int
const  int  * pc  =  &k;         // a pointer to a const int
++ (*pc) ;                       // illegal:  int  *pc is const
++ pc ;                          // ok:  increments pointer to pc
const int*  const cpc  =  &k ;   // a const pointer to a const int
++  (*cpc) ;                     // illegal:  int  *cpc is const
++ cpc ;                         // illegal:  pointer cpc is const
```

لاحظ أن العامل المرجعي * يمكن استخدامه في الاعلان مع أو بدون مسافة في كلا الجانبين . لهذا فإن الاعلانات الثلاثة التالية متكافئة :

```
int*  p;        // indicates that p has type int* (pointer to int)
int  *  p;      // style sometimes used for clarity
int  *p;        // old C style
```

12.6 مصفوفات المؤشرات والمؤشرات للمصفوفات

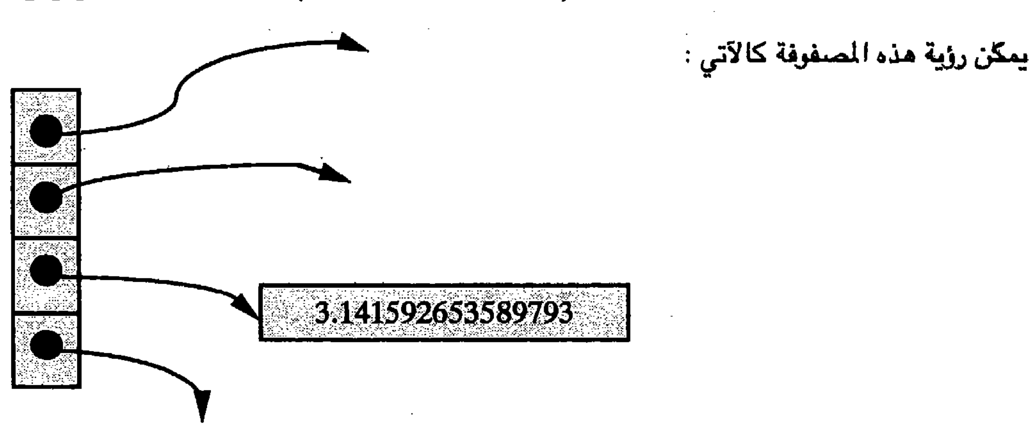
عناصـــر المصفوفـــة يمـكن أن تكـون مؤشـــرات. المصفوفة التالية هي مصفوفة من 4 مؤشرات من النوع double:

```
double*  p [4]
```

عناصرها يمكن تخصيصها مثل أي مؤشر آخر

```
p [2]  =  new double (3.141592653589793) ;
```

يمكن رؤية هذه المصفوفة كالآتي :



المثال التالي يوضح تطبيق مفيد لمصفوفات المؤشر . المثال يوضح كيف ترتب قائمة بطريقة غير مباشرة بتغيير المؤشرات للعناصر بدلاً من تحريك العناصر نفسها. يكافئ هذا الترتيب الفقاعي الغير مباشر المبين في المسألة 12.5 .

مثال 14.6 : الترتيب الفقاعي الغير مباشر

```
// The Indirect Bubble Sort sorts the pointer array:
void sort (float*  p [ ],  int  n)
{
     float*  temp ;
     for  (int  i  =  1 ;   i  <  n ;  i ++)
          for  (int  j  =  0 ;   j  <  n-1 ;   j++)
               if  (*p [j]  >  *p [j+1])  {
```

```
                temp = p [j] ;
                p [j] = p [j+1] ;
                p [j+1] = temp;
            }
}
```

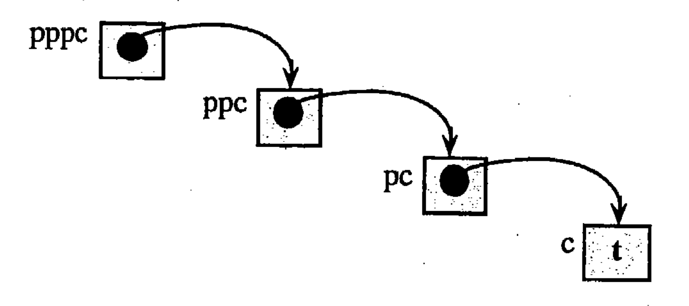
في كل دورة للحلقة الداخلية إذا كانت floats للمؤشرات المتجاورة في غير الترتيب عندئذ تستبدل المؤشرات.

13.6 مؤشرات لمؤشرات

يمكن أن يشير مؤشر إلى مؤشر آخر . على سبيل المثال

```
char c = 't';
char* pc = &c;
char** ppc = &pc;
char*** pppc = &ppc;
***pppc = 'w';              // changes the value of c to 'w'
```

يمكن أن نتصور هذه المتغيرات كالآتي :



التنسيب ' pppc = ' w*** تشير لمحتويات العنوان pc المشار له بالعنوان ppc المشار إليه بالعنوان pppc .

14.6 المؤشرات للدوال

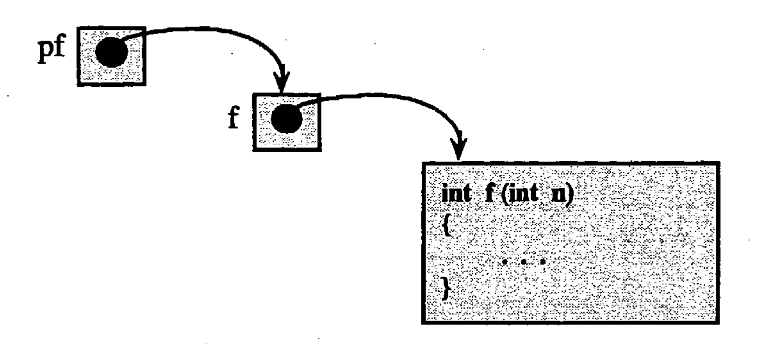
مثل اسم المصفوفة اسم الدالة هو في الحقيقة مؤشر ثابت. يمكن أن نفكر في قميته كعنوان الشفرة التي تنفذ الدالة.

مؤشر لدالة هو ببساطة مؤشر قيمته عنوان اسم الدالة. حيث أن هذا الاسم هو نفسه مؤشر لذلك فإن مؤشر لدالة هو مجرد مؤشر لمؤشر ثابت .

كمثال

```
int  f (int) ;              // declares the function  f
int  (*pf) (int) ;          // declares the function pointer pf
pf  =  &f ;                 // assigns the address of  f  to  pf
```

يمكن أن نتصور مؤشر الدالة مثل



إن قيمة مؤشرات الدالة أنها تسمح لنا بتعريف دوال الدوال. يتم ذلك بإمرار مؤشر دالة كمعامل لدالة أخرى .

مثال 15.6 : جمع الدالة

الدالة () sum لها معاملات : مؤشر الدالة pf والعدد الصحيح n :

```
int sum  (int  (*)  (int) , int) ;
int square  (int) ;
int cube  (int) ;

main  ()
{
     cout  <<  sum  (square,  4)  <<  endl ;   // 1 + 4  +  9  +  16
     cout  <<  sum  (cube,  4)  <<  endl ;     // 1  +  8  +  27  +  64
}
```

النداء (square, 4) sum يحسب ويعيد مجموع square [1] + square [2] + square [3] + square [4] حيث أن square [k] تحسب وتعيد k*k فالدالة () sum تعيد 1+4+9+16 = 30 .

ها هو توصيف الدوال والخرج.

```
// Returns the sum  f (0) + f (1) + f (2) + ... + f (n-1) :
int sum  (int  (*pf)  (int  k) ,  int  n)
{
     int  s  =  0 ;
     for  (int  i  =  1;  i  <=  n ;  i++)
          s  +=  (*pf)  (i) ;
     return  s ;
}

int square  (int  k)
{
     return  k*k ;
}

int cube  (int  k)
{
     return  k*k*k ;
}

30
100
```

الدالة () sum تحسب الدالة التي تشير لها pf ، في كل من الأرقام الصحيحة 1 إلى n وتعيد مجموع هذه القيم التي عددها n .

لاحظ أن الاعلان عن معامل مؤشر الدالة pf في قائمة معاملات الدالة () sum يحتاج المتغير الهيكلي k.

15.6 : NUL ، NULL ، و void

الثابت 0 (صفر) له النوع int . ومع ذلك فإن هذا الرمز يمكن أن ينسب لكل الأنواع الاساسية

```
char  c  =  0 ;            // initializes c to the char  '\0'
short  d  =  0 ;           // initializes d to the short int  0
int  n  =  0 ;             // initializes n to the int  0
unsigned  u  =  0 ;        // initializes u to the unsigned int  0
float  x  =  0 ;           // initializes x to the float  0.0
double  z  =  0 ;          // initializes z to the double  0.0
```

في كل حالة يبدأ الهدف بالرقم 0 . في حالة النوع char ، الحرف c يصبح الحرف الصفري '0' أو NUL . انه الحرف الذي شفرته في ASCII هو 0 .

قيم المؤشرات هي عناوين بالذاكرة . هذه العناوين يجب أن تبقى في الجزء بالذاكرة المخصص لعملية التنفيذ باستثناء العنوان 0x0 .

يسمى هذا المؤشر NULL . يطبق نفس الثابت للمؤشرات المستنتجة من أي نوع :

```
char*  pc  =  0 ;              // initializes pc to NULL
short*  pd  =  0 ;             // initializes pd to NULL
int*  pn  =  0 ;               // initializes pn to NULL
unsigned*  pu  =  0 ;          // initializes pu to NULL
float*  px  =  0 ;             // initializes px to NULL
double*  pz  =  0 ;            // initializes pz to NULL
```

المؤشر NULL لا يمكن اعادة المرجعية اليه . هذا خطأ شائع ولكنه جسيم

```
int*  p  =  0 ;
*p  =  22 ;          // ERROR:  cannot dereference the NULL pointer
```

احتياط معتدل هو اختبار المؤشر قبل محاولة اعادة المرجعية له :

```
if  (p)   * p  =  22 ;  //  OK

*p  =  22 ;     // ERROR: cannot dereference the NULL pointer
```

هنا يختبر الشرط (p != NULL). لأن هذا الشرط حقيقياً عندما تكون p غير صفرية بالضبط.

يعبر الاسم void عن نوع اساسي خاص . ليس مثل كل الأنواع الأساسية الأخرى يمكن استخدام void فقط في النوع المستنتج :

```
void  x ;    // ERROR: no object can have type void
void*  p ;   // OK
```

الاستخدام الأكثر شيوعاً للنوع void هو توصيف أن الدالة لا تعيد قيمة :

```
void  swap  (double&,  double&) ;
```

استخدام آخر مختلف لـ void هو اعلان مؤشر لهدف غير معروف النوع :

```
void*  p  =  q ;
```

هذا الاستخدام أكثر شيوعاً في برامج C المنخفضة المستوى والمصممة للسيطرة على الأجهزة والمكونات المتصلة بالحاسب .

أسئلة مراجعة

1.6 كيف يمكنك الوصول إلى عنوان متغير بالذاكرة ؟

2.6 كيف تحصل على محتويات مكان بالذاكرة عنوانه مخزن في متغير مؤشر ؟

3.6 اشرح الفرق بين الاستخدامين التاليين للعامل المرجعي & :

```
int&  r = n;
p = &n;
```

4.6 اشرح الفرق بين الاستخداميين التاليين للعامل الغير مباشر * :

```
Int*  q = r;
n = *p;
```

5.6 ما هو المؤشر المعلق "Dangling Pointer" ؟

6.6 ما هي التوابع المقلقة التي تنجم عن اعادة المرجعية للمؤشر المعلق ؟

7.6 كيف يمكن تجنب هذه التوابع المقلقة ؟

8.6 ما هو الخطأ في الشفرة التالية :

```
int& r = 22;
```

9.6 ما هو الخطأ في الشفرة التالية :

```
int* p = &44 ;
```

10.6 ما هو الخطأ في الشفرة التالية :

```
char c = 'w';
char p = &c;
```

11.6 ما هو الفرق بين " الربط الاستاتيكي " و " الربط الديناميكي " ؟

12.6 ما هو الخطأ في الشفرة التالية :

```
char c = 'w';
char* p = c;
```

13.6 ما هو الخطأ في الشفرة التالية :

```
short a [32] ;
for (int i = 0 ; i < 32 ; i++)
*a++ = i*i ;
```

14.6 حدد القيمة لكل من المتغيرات المشار إليها بعد تنفيذ الشفرة التالية. افترض أن العدد الصحيح يحتل 4 بايت وأن m مخزن بالذاكرة اعتباراً من البايت 0x3fffd00 .

```
int m = 44 ;
int* p = &m ;
int& r = m ;
int n = (*p)++ ;
int* q = p - 1 ;
r = * (--p) + 1 ;
++*q ;
```

a. m

b. n

c. &m

d. *p

e. r

f. *q

15.6 صنــف كــل من الآتي كقيمة يسارية قابلة للتنفيذ أو قيمة يسارية غير قابلة للتنفيذ أو ليست قيمة يسارية

a. double x = 1.23 ;
b. 4.56*x + 7.89
c. const double y = 1.23 ;
d. double a [8] = {0.0} ;
e. a [5]
f. double f ( ) { return 1.23 ; }
g. f (1.23)
h. double& r = x ;
i. double* p = &x ;
j. *p
k. const double* p = &x ;
l. double* const p = &x ;

16.6 ما هو الخطأ في الشفرة التالية :

```
float x = 3.14159;
float* p = &x;
short d = 44;
short* q = &d;
p = q;
```

17.6 ما هو الخطأ بالشفرة التالية :

```
int* p = new int;
int* q = new int;
cout << "p = " << p << ", p+q = " << p + q << endl;
```

18.6 ما هو الشئ الوحيد الذي يمكن أن تفعله بالمؤشر الصفري NULL ؟

19.6 في الاعلان التالي ، اشرح ما هو نوع p واوصف كيف يمكن استخدامها :

```
double **** p.
```

20.6 اذا كانت x لها العنوان 0x3fffd1c ، فما هي القيم التي تأخذها كل من p ، q في كل من ما يأتي:

```
double x = 1.01;
double* p = &x;
double* q = p + 5;
```

21.6 اذا كانت كل من p ، q مؤشرات لـ int و n هو int أي من الآتي يكون مسموحاً به (صحيح) :

a. p + q
b. p - q
c. p + n
d. p - n
e. n + p
f. n - q

22.6 ماذا يعني القول أن المصفوفة هي في الحقيقة مؤشر ثابت ؟

23.6 كيف يمكن للدالة الوصول لكل عنصر في المصفوفة عندما يمرر لها عنوان العنصر الأول فقط ؟

24.6 اشرح لماذا أن الثلاث حالات التالية حقيقية للمصفوفة a والعدد الصحيح i :

```
a [i]  ==  * (a + i) ;
* (a + i)  ==  i [a] ;
a [i]  ==  i [a] ;
```

25.6 اشرح الفرق بين الاعلانين التاليين :

```
double  *  f  () ;
double  (* f) () ;
```

26.6 اكتب الاعلان لكل من ما يأتي :

أ - مصفوفة من 8 عناصر للأعداد الحقيقية (floats) .
ب - مصفوفة من 8 مؤشرات للأرقام الحقيقية (float).
جـ - مؤشر لمصفوفة من 8 أرقام حقيقية (float) .
د - مؤشر لمصفوفة من 8 مؤشرات إلى أرقام حقيقية (float).
هـ - دالة تعيد رقم حقيقي (float).
و - مؤشر لدالة تعيد رقم حقيقي (float).
ز - مؤشر لدالة تعيد مؤشر إلى رقم حقيقي (float) .
ح - مؤشر لدالة تعيد مؤشر إلى رقم حقيقي (float) .

مسائل برمجة محلولة

27.6 اكتب دالة تستخدم المؤشرات لنسخ مصفوفة من النوع double.

تستخدم الدالة () copy العامل new لتخصيص مصفوفة من العدد n من العناصر والنوع doubles . يحتوي المؤشر p على عنوان العنصر الأول لهذه المصفوفة الجديدة حتى يمكن استخدامه لاسم المصفوفة كما في p [i] . وبعد نسخ عناصر المصفوفة a في المصفوفة الجديدة يعاد p بالدالة .

```
double*  copy  (double  a [] ,   int  n )
{
     double*  p   =  new  double  [n] ;
     for  (int   i  =  0 ;   i  <  n ;  i++)
          p [i]   =  a [i] ;
     return  p ;
}
```

```
void print (double [ ] ,  int) ;

main ( )
{
    double a [8] = {22.2, 33.3, 44.4, 55.5, 66.6, 77.7, 88.8, 99.9} ;
    print (a, 8) ;
    double* b = copy (a, 8) ;
    a [2] = a [4] = 11.1 ;
    print (a, 8) ;
    print (b, 8) ;
}
```

```
22.2,  33.3,  44.4,  55.5,  66.6,  77.7,  88.8,  99.9
22.2   33.3,  11.1,  55.5,  11.1,  77.7,  88.8,  99.9
22.2,  33.3,  44.4,  55.5,  66.6,  77.7,  88.8,  99.9
```

في هذا التنفيذ نبدأ a كمصفوفة مكونة من 8 عناصر والنوع double . نستخدم الدالة () print لفحص محتويات a . استدعينا الدالة () copy وقيمة عائدها نسبت للمؤشر b الذي يخدم بعد ذلك كاسم للمصفوفة الجديدة . قبل طباعة b ، نغير قيمة عنصرين من a لنختبر أن b ليست نفس المصفوفة مثل a كما يؤكد الندائين الأخيرين () print .

28.6 اكتب دالة تستخدم المؤشرات للبحث عن عنوان رقم صحيح معطي في مصفوفة معطاة . إذا وجد الرقم المعطي ، تعيد الدالة عنوان وإلا فتعيد NULL.

نستخدم الحلقة for للتحرك عبر المصفوفة . إذا وجد الهدف عند [a [i فان عنوانه [i] a& يعاد . وإلا فيعاد NULL.

```
int* location (int  a [] ,  int n ,  int  target)
{
    for  (int i = 0 ; i < n ; i++)
        if (a [i] == target)  return &a [i] ;
    retrun NULL ;
}
```

البرنامج الاختباري يستدعي الدالة ويخزن عنوان العودة في الموشر p إذا كان غير الصفر (أي ليس NULL) عندها يطبع العنوان وقيمة الرقم الصحيح int .

```
main ()
{
    int a [8] = {22, 33, 44, 55, 66, 77, 88, 99}, * p, n;
    do {
        cin >> n ;
        if (p = location (a, 8, n)) cout << p << ", " << *p << endl;
        else cout << n << " was not found. \n" ;
    } while (n > 0);
}
```

```
44
0x3fffcc4, 44
50
50 was not found.
99
0x3fffcd8, 99
90
90 was not found.
0
0 was not found.
```

29.6 اكتب دالة اذا ارسلنا لها مصفوفة مكونة من عدد n من المؤشرات إلى أرقام حقيقية float تعيد مصفوفة جديدة تحتوي على عدد n من القيم الحقيقية float .

نستخدم الحلقة for لنتحرك داخل المصفوفة إلى أن تشير p للهدف :

```
float* duplicate (float* p [], int n)
{
    float* const b = new float [n];
    for (int i = 0; i < n; i ++, q ++)
        b [i] = *p [i];
    return b;
}

void print (float [ ], int);
void print (float* [ ], int);
```

```
main ( )
{
    float a [8] = {44.4, 77.7, 22.2, 88.8, 66.6, 33.3, 99.9, 55.5} ;
    print (a, 8) ;
    float* p [8] ;
    for (int i = 0 ; i < 8 ; i++)
        p [i] = &a [i] ; // p [i] points to a [i]
    print (p, 8) ;
    float* const b = duplicate (p, 8) ;
    print (b, 8);
}
```

```
44.4, 77.7 22.2, 88.8, 66.6, 33.3, 99.9, 55.5
44.4, 77.7 22.2, 88.8, 66.6, 33.3, 99.9, 55.5
44.4, 77.7 22.2, 88.8, 66.6, 33.3, 99.9, 55.5
```

30.6 نفذ دالة للتكامل بطريقة جمع ريمان "Riemann sums" استخدم العلاقة

$$\int_a^b f(x)\,dx = \left(\sum_{i=1}^{n} f(a+ih)\right) h$$

هذه الدالة المسماة riemann () تشابه الدالة sum () في المثال 15.6 معاملها الأول هو مؤشر لدالة لها معامل واحد double وتعيد double . في هذا التنفيذ الاختباري نمرر لها (مؤشر لـ) الدالة cube () . المعاملات الثلاث الأخرى هي الحدود a ، b في الفترة [a, b] التي يجري عليها التكامل والعدد n للمسافات الجزئية المستخدمة في الجمع . الجمع الحقيقي لـ Riemann هو مجموع المساحات لعدد n من المستطيلات المرتكزة على هذه المسافات الجزئية والتي ارتفاعاتها معطاة بالدالة المراد تكاملها .

```
double riemann (double (*) (double), double, double, int) ;
double cube (double) ;

main ()
{
    cout << riemann (cube, 0,2,10) << endl ;
    cout << riemann (cube, 0,2,100) << endl ;
    cout << riemann (cube, 0,2,1000) << endl ;
    cout << riemann (cube, 0,2,10000) << endl ;
}
```

```
// Returns [f (a) * h + f (a+h)*h + f (a+2h)*h + . . . + f (b-h)*h],
// where h = (b-a)/n :
double riemann (double (*pf) (double t), double a, double b, int n)
{
    double s = 0, h = (b-a)/n, x;
    int i;
    for (x = a, i = 0; i < n; x += h, i++)
        s += (*ph) (x);
    return s*h;
}

double cube (double t)
{
return t*t*t
}
```

```
3.24
3.9204
3.992
3.9992
```

في هذا التنفيذ التجريبي نكامل الدالة $y = x^3$ على الفترة [0, 2] . بواسطة الحساب الأولي قيمة هذا التكامل هي 4.0 . النداء (riemann (cube, 0, 2, 10 يقرب التكامل باستخدام عشر فترات جزئية لتحصل على 3.24 . أما النداء (riemann (cube, 0, 2, 100 يقرب التكامل باستخدام مائة فترة جزئية ليعطي 3.9204 هذا الجمع يقترب أكثر فأكثر للنهاية 4.0 كلما زادت n . مع 10000 فترة جزئية مجموع ريمان يعطي 3.9992 .

لاحظ أن الفرق الجوهري الوحيد بين هذه الدالة ( ) riemann والدالة () sum في المثال 15.6 هو أن المجموع مضروب في عرض الفترة الجزئية h قبل إعادته .

31.6 اكتب دالة تعيد المشتقة العددية لدالة رياضية f عند نقطة معطاة x . مستخدماً سماح معطى h .

استخدم العلاقة :

$$f'(x) = \frac{f(x+h) - f(x-h)}{2h}$$

هذه الدالة () derivative تشابه الدالة () sum بالمثال 15.6 عدا أنها تستخدم المعادلة بدلاً من المشتقة العددية. لها ثلاث معاملات: مؤشر للدالة f ، قيمة x ، والسماح h . في هذا الاختبار للتنفيذ نرسل لها (مؤشرات إلى الدالة () cube والدالة () sqrt.

```
#include <iostream.h>
#include <math.h>

double derivative (double (*) (double) , double, double) ;
double cube (double) ;

main ()
{
        cout << derivative (cube, 1, 0.1) << endl ;
        cout << derivative (cube, 1, 0.01) << endl ;
        cout << derivative (cube, 1, 0.001) << endl ;
        cout << derivative (sqrt, 1, 0.1) << endl ;
        cout << derivative (sqrt, 1, 0.01) << endl ;
        cout << derivative (sqrt, 1, 0.001) << endl ;
}

// Returns an approximation to the derivative f' (x) :
double derivative (double (*pf) (double t) , double x , double h)
{
    return ( (*pf) (x+h) - (*ph) (x-h) ) / (2*h) ;
}

double cube (double t)
{
    return t*t*t
}
```

```
3.01
3.0001
3
0.500628
0.500006
0.5
```

المشتقة للدالة () cube x^3 هي $3x^2$ وقيمتها عندما $x = 1$ هي 3 . ولهذا فالمشتقة العددية يجب أن تقترب من 3.0 عندما تكون h كبيرة. بالمثل المشتقة للدالة () sqrt $\sqrt{x}$ هـو $1/(2\sqrt{x})$ ، وقيمتها عند $x = 1$ هو 1/2 . ولذلك قيمة مشتقتها العددية يجب أن تقترب من 0.5 بقيمة كبيرة لـ h.

32.6 اكتب دالة ترسل لها مصفوفة من عدد n مؤشر إلى floats وتعيد مؤشر لأكبر رقم في تلك المجموعة.

يستخدم المؤشر pmax لتحديد الرقم الأكبر ، فهو يبدأ بالقيمة p [0] التي تشير إلى أول float . ثم بداخل الحلقة for الرقم الذي يشير إليه p [i] يقارن بالرقم الذي يشير اليه pmax وتحدد قيمة pmax لتشير للرقم الأكبر عندما يكتشف . لهذا عندما تنتهي الحلقة فإن pmax يشير إلى أكبر رقم حقيقي float بالمجموعة .

```
float* max  (float* p [] ,  int n)
{
     float* pmax = p [0] ;
     for (int i = 1 , i < n , i++)
          if (*p [i] > *pmax) pmax = [i] ;
     return pmax ;
}

void print (float [ ] , int) ;
void print (float* [ ] , int) ;

main ()
{
     float a [8] = {44.4, 77.7, 22.2, 88.8, 66.6, 33.3, 99.9, 55.5};
     print (a, 8) ;
     float* p [8] ;
     for (int i = 0; i < 8 ; i++)
          p [i] = &a [i] ; // p [i] points to a [i]
     print (p, 8) ;
     float* m = max (p, 8) ;
     cout << m << ", " << *m << endl ;
}
```

```
44.4, 77.7, 22.2, 88.8, 66.6, 33.3, 99.9, 55.5
44.4, 77.7, 22.2, 88.8, 66.6, 33.3, 99.9, 55.5
0x3fffcd4, 99.9
```

لدينا هنا دالتان للطبع print () محملتان. واحدة لطبع مصفوفة المؤشرات والثانية لطبع الأرقام التي تشير إليها. بعد البدأ وطبع المصفوفة a فإننا نعرف المصفوفة p ونبدأ قيم عناصرها لتشير إلى عناصر a . النداء print (p, 8) يثبت أن p تحقق وصول غير مباشر إلى a . أخيراً يعلن عن المؤشر n ويبدأ بالعنوان العائد من الدالة max () . الخرج الأخير يؤكد أن m فعلاً تشير إلى أكبر رقم بين الأرقام التي وصلنا إليها بواسطة p .

مسائل إضافية

33.6 اكتب الدالة التالية التي نرسل لها مصفوفة من n مؤشر إلى أعداد حقيقية floats وتعيد مصفوفة جديدة تحتوي على هذه الأرقام بالترتيب المعاكس.

```
float* mirror (float* p [ ], int n)
```

34.6 اكتب الدالة التالية التي تعيد عدد البايتات التي يجب أن تزاد بها s قبل أن تشير i إلى الحرف الصفري ' \0 ' :

```
unsigned len (const char* s)
```

35.6 اكتب الدالة التالية والتي تنسخ أول عدد n بايتات ابتداءاً من s2* إلى البايتات التي بـ s1* ، حيث n هو عدد البايتات التي يجب أن تزاد اليها s2 قبل أن تشير إلى الحرف الصفري ' \0 ' :

```
void cpy (char* s1 , const char* s2)
```

36.6 اكتب الدالة التالية التي تنسخ أول عدد n من البايتات ابتداءاً من s2* من البايتات ابتداءاً من مكان أول وجود للحرف الصفري ' \0 ' بعد s1* ، حيث n هو عدد البايتات التي يجب زيادتها على s2 قبل أن تشير إلى الحرف الصفري ' \0 ' :

```
void cat (char* s1 , const char* s2)
```

37.6 اكتب الدالة التالية التي تقارن n من البايتات على الأكثر ابتداءاً بـ s2 بالبايتات المناظرة ابتداءاً بـ s1، حيث n هو عدد البايتات التي يجب أن تزاد بها s2 قبل أن تشير إلى الحرف الصفري ' \0 ' . إذا تطابقت كل الـ n بايت ، يجب أن تعيد الدالة 0 . وإلا فتعيد إما 1- أو 1 طبقاً لما إذا كانت البايت من s1 أقل أو أكبر من البايت من s2 في أول عدم تطابق :

```
int cmp (char* s1, char* s2)
```

38.6 اكتب الدالة التالية والتي تفحص الـ n بايت ابتداءاً من s عن الحرف C . حيث n هو عدد البايتات التي يجب أن تزاد بها s قبل أن تشير للحرف الصفري ' \0 ' إذا وجد الحرف يعاد مؤشر له ; وإلا يعاد NULL :

```
char chr (char* s , char* c)
```

39.6 اكتب الدالة التالية التي تعيد مجموع الأرقام المشار لها بالعدد n من المؤشرات الأولى في المصفوفة p :

```
void sum (float* p [ ] , int n]
```

40.6 اكتب الدالة التالية التي تعيد اشارة كل رقم سالب مشار اليه بأول n من المؤشرات في المصفوفة p :

```
void abs (float*  p [ ] ,  int  n)
```

41.6 اكتب الدالة التالية والتي ترتب بطريقة غير مباشرة الاعداد المشار إليها بأول عدد n من المؤشرات في المصفوفة p عن طريق إعادة ترتيب المؤشرات

```
void sort (float*  p [ ] ,   int   n)
```

42.6 طبق الاختيار الغير مباشر للترتيب باستخدام مصفوفة مؤشرات (انظر المسألة 35.5) .

43.6 طبق الادخال الغير مباشر للترتيب (انظر المسألة 36.5).

44.6 نفذ التوزيع العشوائي الغير مباشر (انظر المسألة 15.5) .

45.6 اعد كتابة الدالة ( ) sum (مثال 15.6) حتى تطبق على الدوال مع اعادة النوع double بدلاً من int . ثم اختبرها على الدالة ( ) sqrt (المعرفة في <math.h>) والدالة العكسية.

46.6 طبق الدالة () riemann (المسألة 30.6) على الدوال التالية والمعرفة في <math.h> :

a. sqrt () , on the interval [1, 4] ;

b. cos () , on the interval [0, $\pi/2$] ;

c. exp (), on the interval [0, 1] ;

d. log (), on the interval [1, e] .

47.6 طبق دالة المشتقة () derivative (المسألة 31.6) على الدوال التالية والمعرفة في <math.h> :

a. sqrt () , at the point x = 4 ;

b. sin () , at the point x = $\pi/6$;

c. exp (), at the point x = 0 ;

d. log (), at the point x = 1.

48.6 اكتب الدالة التالية والتي تعيد ضرب n من القيم f(n) , ... , f(2) , f (1) (انظر المثال 15.6):

```
int product  (int  (*pf)  (int  k) ,   int  n)
```

49.6 استخدم طريقة التقسيم الثنائي Bisection Method لحل المعادلات مستخدماً الدالة التالية :

```
double root  (double  (*pf)  (double  x) ,  double  a,  double b,  int  n)
```

هنا يشير pf إلي الدالة f التي تحدد المعادلة f (x) = 0 المراد حلها.

الفترة a ، b تحصر الجذر المجهول x (أي $a \le x \le b$) و n هو عدد المحاولات في تلك الفترة . كمثال : النداء (root (square , 1, 2, 100 سوف يعيد الرقم 1.414213562 $(\sqrt{2} =)$ أي الحـل للمعادلـة $x^2 = 2$. طريقة التقسيم الثنائى تعمل بتكرار تنصيف الفترة واستبدالها بالنصف الذي يحتوي على الجذر. تختبر اشارة حاصل الضرب f (a) f(b) لتحدد ما إذا كان الجذر في الفترة [a , b].

50.6 طبق قاعدة شبه المنحرف لتكامل دالة . استخدم الدالة التالية :

```
double trap (double (*pf) (double x), double a, double b, int n)
```

هنا pf يشير إلى الدالة f المراد تكاملها . a و b تحصر الفترة [a, b] والتي خلالها نكامل f و n هو عدد الفترات الجزئية المستخدمة . كمثال النداء (trap (square, 1, 2, 100 سوف يعيد 1.41421 . قاعدة شبـه المنحـرف تعيد مجموع المساحات لـ n من أشباه المنحرفات والتي تقرب المساحة تحت المنحنى للدالة f .

كمثال : إذا كانت n = 5 فإنها تعيد الآتي ، حيث $h = (b - a) / 5$ هو عرض كل شبه منحرف :

$$h/2 \ [f(a) + 2f(a+h) + 2f(a+2h) + 2f(a+3h) + 2f(a+4h) + f(p)]$$

اجابات أسئلة المراجعة

1.6 طبق عامل العنوان & على المتغير x& .

2.6 طبق عامل اعادة المرجعية * على المتغير p* .

3.6 الاعــلان ; int & r = n يــعلن عــن r على أنــها المرجــع (مـرادف) للمتغير n من النوع int . التنسيب ; p = &n ينسب عنوان n للمؤشر p .

4.6 الاعلان ;int* q = p يعلن عن q على أنها مؤشر (عنوان بالذاكرة) يشير لنفس العدد الصحيح int والذي يشير إليه p . والتنسيب n = *p ينسب إلى n العدد الصحيح int الذي يشير إليه p .

5.6 "المؤشر المعلق" هو مؤشر لم يتم بدأ قيمته . انه خطير حيث انه قد يشير إلى ذاكرة غير مخصصة أو ذاكرة لا يمكن الوصول إليها.

6.6 اذا اعيدت المرجعية لمؤشر إلى مكان غير مخصص بالذاكرة فإنه قد يغير قيمة بعض المتغيرات الغير معروفة. إذا اعيدت المرجعية لمؤشر إلى مكان غير ممكن الوصول إليه بالذاكرة فإنه يحتمل فشل البرنامج (crash) (ينتهي فجأة) .

7.6 ابدأ قيمة المؤشرات عند اعلانها.

8.6 لا يمكنك الرجوع إلى ثابت عنوانه لا يمكن الوصول إليه.

9.6 عامل المرجعية & لا يجوز تطبيقه على ثابت .

10.6 المتغير p له النوع char ولكن التعبير &c له النوع مؤشر إلى char لبدأ p إلى &c , p يجب أن يعلن عنه على النوع *char .

11.6 الربط الاستاتيكي يكون عند تخصيص الذاكرة في وقت الترجمة . كما في اعلان المصفوفة :

```
double  a [400] ;
```

الربط الديناميكي يكون عند تخصيص الذاكرة عند تنفيذ البرنامج باستخدام العامل new :

```
double*  p;
p  =  new double  [400] ;
```

12.6 المتغير p له النوع *char بينما التعبير c له النوع char . لبدء p ليكون c فإن p لابد أن يكون لها نفس النوع مثل c . اما كلاهما char أو كلاهما *char .

13.6 المشكلة الوحيدة أن اسم المصفوفة a هو مؤشر ثابت ولذا لا يمكن ازادتها. الشفرة التالية المعدلة لا بأس بها :

```
short  a [32] ;
short*  p  =  a ;
for  (int  i  =  0 ;   i  <  32 ;  i ++)
*p ++  =  i*i ;
```

14.6

a. m = 46

b. n = 44

c. &m = 0x3fffd00

d. *p = 46

e. *r = 46

f. *q = 46

15.6

a. قيمة يسارية مخففة.

b. ليست يسارية.

c. قيمة يسارية غير مخففة.

d. قيمة يسارية غير مخففة.

e. قيمة يسارية مخففة.

f. قيمة يسارية غير مخففة.

g. قيمة يسارية مخففة اذا كان نوع العائد مرجع غير محلي، وإلا فانها ليست قيمة يسارية.

h. قيمة يسارية مخففة.

i. قيمة يسارية مخففة.

j. قيمة يسارية الا إذا اشارت p إلى ثابت . في هذه الحالة p* تكون قيمة يسارية غير مخففة.

k. قيمة يسارية مخففة.

l. قيمة يسارية غير مخففة.

16.6 المؤشرات p ، q مختلفا النوع. p مؤشر لـ float بينما q مؤشر لـ short . من الخطأ أن تنسب العنوان في نوع مؤشر لمؤشر آخر مختلف النوع.

17.6 من الخطأ اضافة مؤشرين.

18.6 اختبرها لترى ما اذا كانت NULL على الأخص لا تعيد المرجعية لها .

19.6 p هو مؤشر لمؤشر لمؤشر إلى double . يمكن استخدامها لتعبر عن مصفوفة رباعية الأبعاد .

20.6 قيمة p هي نفسها عنوان a : 0x3fffd1c . قيمة q تعتمد على sizeof (double) . اذا احتلت أهداف من النوع double ثماني بايتات فإن ازاحة قدرها 8 × 5 = 40 تضاف إلى p لتعطي q القيمة الستعشرية 0x3fffd44.

21.6 التعبير الوحيد بين التعبيرات الستة الغير صحيحة هي p + q و n - q .

22.6 اسم المصفوفة هو متغير يحتوي على عنوان العنصر الأول من المصفوفة. هذا العنوان لا يمكن تغييره ولهذا فإن اسم المصفوفة في الحقيقة هو مؤشر ثابت.

23.6 في الشفرة التالية التي تجمع كل عناصر المصفوفة a. كل زيادة للمؤشر تحدد العنصر التالي.

```
const SIZE = 3 ;
short a [SIZE] = {22, 33, 44} ;
short* end = a + SIZE; // adds SIZE* sizeof (short) = 6 to a
for (short* p = a ; p < end ; p++)
    sum += *p ;
```

24.6 القيمة a [i] المعادة بواسطة الرمز الفرعي [ ] . هي القيمة المخزنة بالعنوان المحسوب بالتعبير a + i . في هذا التعبير a هي مؤشر للنوع الاساسي t و i هو int ولذلك فإن الازاحة (t) i*sizeof تضاف إلى عنوان a . نفس الحساب يتم من خلال التعبير a + i والذي يستخدم في i [a] .

25.6 الاعـــلان f () *double يعلــن عــن f لتكــون الدالــة التـي تعـيد مـؤشـر إلى double . الاعلان double (*f) ( ); يعلن عن f* ليكون مؤشر إلى دالة تعيد double .

26.6

a. float a [8] ;

b. float* a [8] ;

c. float (* a) [8] ;

d. float* (* a) [8] ;

e. float f () ;

f. float* f () ;

g. float (* f) () ;

h. float* (* f) () ;

الفصل السابع

السلاسل
Strings

7

1.7 مقدمة

السلسلة (أيضاً تسمى سلسلة حروف) هو تتابع من الحروف المتجاورة في الذاكرة تنتهي بالحرف الصفري NUL ' \0 ' . يتم التعامل مع السلاسل بواسطة متغيرات من النوع *char (مؤشر إلى حرف) وعلى سبيل المثال إذا كانت s لها النوع *char فإن ; cout << s << endl

ستطبع كل الحروف المخزنة بالذاكرة ابتداءاً من عنوان s وانتهاءاً بأول مقابلة للحرف الصفري NUL ' \0 ' .

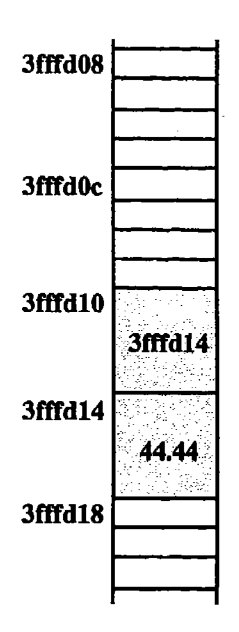
ملف الرأس <string.h> يحتوي على عدد كبير من الدوال للتعامل مع السلاسل . كمثال : النداء strlen (s) يعيد عدد الحروف بالسلسلة s ولا يتضمن حرف الانهاء NUL . كل هذه الدوال تعلن عن ثوابت السلسلة كمؤشرات إلى char. ولهذا قبل دراسة عمليات السلاسل هذه نحتاج إلى مراجعة للمؤشرات.

2.7 مراجعة للمؤشرات

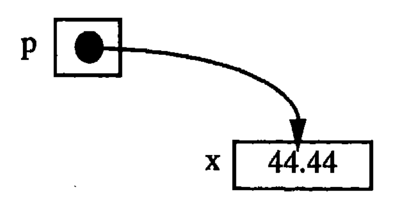
المؤشر هو عنوان بالذاكرة . كمثال الاعلان التالي يحدد x على أنهامتغير حقيقي float يحتوي على القيمة 44.44 و p على أنها مؤشر يحتوي على عنوان x :

```
float x = 44.44 ;
float* p = &x ;
```

اذا تصورنا الذاكرة على انها تتابع للبايتات بالعنوان الستعشري . عندئذ يمكن تصور x و p كالتالي :



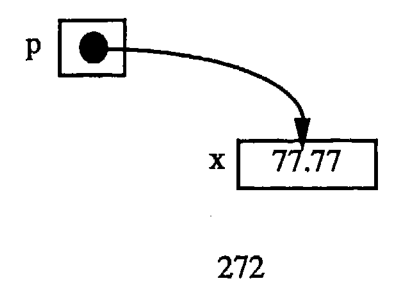
يوضــح هــذا أن x مـخزنـة بالعنوان 3fffd14 و p مـخزنة بالعنوان 3fffd10 . المتـغـير x له القيـمة الحقيقية 44.44 والمتغير p يحتوي على قيمة العنوان 3fffd14 . قيمة p هي العنوان لـ 3fffd14 : x . عادة ما تعبر عن هذه العلاقة بالتخطيط التالي :



هذان المستطيلان ، أحدهمـا معنون p والآخـر مـعنون x . تمثل المسـتطيـلات أمـاكن تخـزين بالذاكـرة . المتغير p يشـير إلى المتـغير x . يمكن الوصــول إلى x من خـلال المؤشـر p عن طريق عـامل إعـادة المرجـعية * . الجملة التالية:

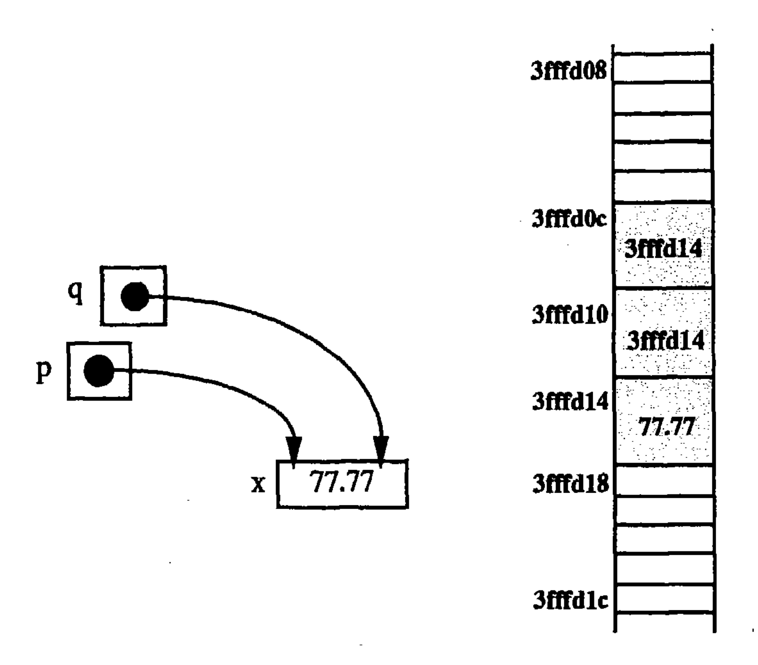
```
*p = 77.77
```

تغير قيمة x إلى 77.77



يمكن أن يكون لدينا أكثر من مؤشر كلها يشير إلى نفس الهدف

```
float* q = &x;
```



الآن p* ، q* و x كلها اسماء لنفـس الـهدف الذي عنوانه 3fffd14 وقيمته الحالية 77.77 . يبين ذلك أن q لها العنوان 3fffd0c . القيمة المخزنة في q هي العنوان 3fffd14 للمتغير x .

المثال التالي يتبع هذه التعاريف في محطة التشغيل UNIX . لاحظ أنه كما تبين هذه الأشكال الذاكرة مخصصة بترتيب تنازلي . الهدف الأول x ، مخزن بالعنوان 3fffd14 ، ويشغل البايتات 3fffd14-3fffd17. الهدف الثاني p مخزن بالعنوان 3fffd10.

مثال 1.7 متابعة المؤشرات

يعرف هذا البرنامج متغير حقيقي x ، مؤشران إلى حقيقي p ، q ، ثم يطبع قيمهما وعناوينهما . ايضاً تطبع قيم الأهداف التي يشير اليها المؤشر :

```
main ()
{
    float x = 44.44 ;
    cout << " x = " << x << endl ;
    cout << "\t&x = " << &x << endl ;    // prints address of x
    float* p = &x ;
    cout << "\np = " << p << endl ;
    cout << "\t&p = " << &p << endl ; // prints address of p
```

```
        cout << "\t*p = " << *p << endl ;     // prints object p points to
        *p = 77.77 ;
        cout << "\np = " << p << endl ;
        cout << "\t&p = " << &p << endl;
        cout << "\t*p = " << *p << endl ;
        cout << "x = " << x << endl;
        cout << "\t&x = " << &x << endl ;
        float* q = &x ;
        cout << "\nq = " << q << endl ;
        cout << "\t&q = " << &q << endl ;
        cout << "\t*q = " << *q << endl ;
        cout << "x = " << x << endl ;
        cout << "\t&x = " << &x << endl ;
}
```

```
x = 44.44          &x = 0x3fffd14

p = 0x3fffd14      &p = 0x3fffd10     *p = 44.44

p = 0x3fffd14      &p = 0x3fffd10     *p = 77.77
x = 77.77          &x = 0x3fffd14

q = 0x3fffd14      &q = 0x3fffd0c     *q = 77.77
x = 77.77          &x = 0x3fffd14
```

لاحظ كيف أخرجت قيم العناوين : 0x3fffd14 هو العدد الستعشري 3fffd14 . التقديم 0x هو الرمز المعتاد للدلالة على أن العدد يتبع النظام الستعشري . رغم أنه ليست هناك حاجة لعمل ذلك ، يمكن للشخص حساب القيمة العشرية المناظرة ، اعتبار أن ' a ' هي الستعشري للرقم العشري 10 ، ' b ' هو 11 ، ' c ' هو 12 ، ' d ' يكون 13 ، ' e ' يكون 14 و ' f ' يكون 15 :

$$0x3fffd14 = 3 \times 16^6 + 15 \times 16^5 + 15 \times 16^4 + 15 \times 16\ 3 + 15 \times 16^3$$
$$+ 13 \times 16^2 + 1 \times 16^1 + 4 \times 16^0 = 67,108,116$$

لهذا في هذا التنفيذ تخزن x فعلياً في 4 بايت مرقمة 67,108,116-67,108,119 تلك هي عناوين ظاهرية في محطة التشغيل UNIX بها ذاكرة 20 مليون بايت. في حاسب شخصي pc يعمل على Dos مع 4 مليون بايت ذاكرة. x و p و q تكون مخزنة بالعناوين :0x23ec و 0x23dc و 0x23be .

إذا كانــت p مـؤشــر ، فـإن النداء p* >> cout دائماً يطبع قيمة الهدف الذي يشير إليه p والنداء p >> cout عادة يطبع قيمة العنوان المخزن في p . استثناء هام لهذه القاعدة الثانية هي عند اعلان p على أنها من النوع *char .

3.7 السلاسل

في ++C السلسلة هي مصفوفة حروف لها الخواص الاساسية التالية :

- يتم إلحاق عنصر إضـافي لنهاية المصفوفة وقيمته تحدد بالحرف الصفري '0\' ' NUL . يعني هذا أن العدد الكلي للحروف بالمصفوفة دائماً يزيد بواحد على طول السلسلة .
- يمكن بدأ السلسلة بحروف مثل :

```
char str [ ] = "Bjarne" ;
```

لاحظ أن هذه المصفوفة بها العناصر '0\' و 'B' , 'j' , 'a' , 'r' , 'n' , 'e' .

- يمكن اخراج السلسلة بالكامل كهدف واحد مثل

```
cout << str ;
```

سينسخ النظام الحروف من str إلى cout حتى يصل إلى الحرف '0\' .

- السلسلة بكاملها يمكن ادخالها كهدف واحد مثل

```
cin >> buffer ;
```

سـينسخ النظـام الحـروف من cin إلـى buffer حـتى يصل إلي الحـرف الفـارغ (مسـافـة) . يجب على المســتخـدم أن يتـأكـد مــن تعريف العـازل buffer على أنه سلسلة حروف كـافية الطول لاستيعاب الدخل.

- يمكن اسـتـخـدام الدوال المعلنة بملف الرأس <string.h> للتعامل مع السلاسل. تتضمن ذلك دالة طول السلسلة () strlen ودالتي نسخ السلسلة () strcpy و () strncpy . ودوال الحاق السلاسل () strcat ، () strncat و دوال مقارنة السلاسل () strncmp ، () strcmp. ودالة استخراج المقاطع () strtok . هذه الدوال موصوفة في الجزء 8.7 .

مثال 2.7 تنتهي السلاسل بالحرف الصفري NUL

هذا البرنامج التوضيحي البسيط يبين أن الحرف الصفري NUL '0\' اضيف إلى آخر السلسلة :

```
main ()
{
    char s [] = "ABCD" ;
    for (int i = 0 ;  i < 5 ; i++)
        cout << "s [" << i << "] = '" << s [i] << "' \n" ;
}
```

```
s [0]  =  'A'
s [1]  =  'B'
s [2]  =  'C'
s [3]  =  'D'
s [4]  =  ''
```

عند ارسال الحرف الصفري Nul إلى cout لا يطبع شيء ولا حتى خانة فارغة . نرى ذلك عن طريق طبع " ، " مباشرة قبل الحرف وأخرى بعده مباشرة .

4.7 ادخالل وإخراج السلاسل I/O

ادخال وإخراج السلاسل يتم بعدة طرق في برامج ++C . الطريقة المثلى بواسطة عامل طبقة السلسلة كما هو موضح بالفصل العاشر . مزيد من الطرق المباشرة نقدمها في هذا الجزء.

مثال 3.7 الدخل والخرج العادي للسلاسل

هذا البرنامج يقرأ كلمات في مخزن مؤقت (عازل) buffer سعته تسعة وسبعون حرفاً :

```
main ()
{
    char word [80] ;
    do {
        cin >> word ;
        if (*word) cout << "\t\" " << word << "\ "\n" ;
    } while (*word) ;
}
```

```
Today' s date is March 12, 1996.
"Today' s "
"date"
" is "
"March"
" 12, "
"1996. "
Tomorrow is Monday.
"Tomorrow"
"is"
"Monday. "
^D
```

في هذا التنفيذ الحلقة while تكررت عشر مرات : واحدة لكل كلمة أدخلت (بما فيها control-D التي أوقفت الحلقة) . كل كلمة في مجرى الدخل cin تم ترديدها في مجرى الخرج cout . لاحظ أن مجرى الخرج لم نتخلص منه إلى أن يصل مجرى الدخل إلى علامة نهاية السطر.

كل سلسلة طبعت بالعلامة " في كل جانب . هذا الحرف لابد من الإشارة اليه عن طريق الحرفين "\ بداخل حروف السلسلة.

التعبير word* يحكم الحلقة إنه الحرف الأول في السلسلة . تكون غير صفرية (أي "حقيقة") طالما السلسلة word تحتوي على سلسلة أطول من الصفر. السلسلة التي طولها صفر يطلق عليها empty string السلسلة الخالية . وتحتوي على الحرف الصفري NUL ، '\0' في أول عنصر به . الضغط على control-D في UNIX أو ماكينتوش (في Dos نضغط contro-z أو VAX/VMS) يرسل حرف نهاية الملف من cin . يحمل هذا السلسلة الخالية في word ويحدد word* (التي هي نفسها [0] word) إلى '\0' ويوقف الحلقة . السطر الأخير في الخرج يظهر فقط لترديد الأمر control-D .

لاحظ أن علامات التنقيط punctuation (' ، , ، : ، إلخ) ليست بالسلاسل. الحلقة do في المثال 3.7 يمكن استبدلها بـ :

```
cin >> word
while (*word) {
    cout << "\t\" " << word << "\"\n";
    cin >> word;
}
```

عند الضغط على control-D ، النداء cin >> word ينسب السلسلة الخالية إلى word .

المثال 3.7 والمثال 1.7 يوضحان خاصية مهمة : عامل الخرج >> يتصرف بصورة مختلفة مع المؤشرات من النوع *char عن أنواع المؤشرات الأخرى . مع المؤشر *char يخرج بالعامل كل سلسلة الحروف التي يشير لها المؤشر . ولكن مع أي نوع آخر للمؤشر ، فإن العامل ببساطة يخرج عنوان المؤشر .

5.7 بعض أعضاء الدوال cin

هدف مجرى بالدخل cin يتضمن دوال الادخال ()cin. get ، ()cin. getline ، ()cin. ignore ، ()cin. putback ، ()cin. peek . كل من أسماء هذه الدوال يتضمن المقدمة ".cin" لأنهم دوال أعضاء للهدف cin . هذا المبدأ للهدف الموجه مشروحة في الفصلين الثامن والثاني عشر.

النداء (cin. getline (str, n يقرأ حتى عدد n من الحروف في str ويهمل الباقي.

مثال 4.7 الدالة ()cin. getline ذات المعاملين

يردد هذا البرنامج الدخل سطر بسطر :

```
main ()
{
    char line [80] ;
    do {
        cin. getline (line, 80) ;
        if (*line) cout << "\t [ " << line << " ] \n " ;
    } while (*line) ;
}
```

```
Once upon a midnight dreary , while I pondered, weak and weary,
    [Once upon a midnight dreary , while I pondered, weak and weary,]
Over many a quaint and curious volume of forgotten lore,
    [Over many a quaint and curious volume of forgotten lore,]
^D
```

لاحظ أن الشرط (line*) سيؤول إلى القيمة "true" بالضبط عندما يحتوي السطر على سلسلة غير خالية. لأنه فقط عندها [0] line تختلف عن الحرف NUL (قيمته في ASCII هي 0) .

النداء (cin. getline (str, n, ch يقرأ كل الدخل حتى أول وجود للحرف الناهي ch في str . إذا كان الحرف المحدد ch هو حرف السطر الجديد '\n' فإن ذلك يكافئ (cin. getline (str, n . يتضح ذلك من المثال التالي حيث أن الحرف الناهي هو الفاصلة ' , ' .

مثال 5.7 الدالة cin. getline ()

يردد هذا البرنامج الدخل. عبارة بعبارة

```
main ()
{
    char clause [20] ;
    do {
        cin. getline (clause, 20, ',') ;
        if (*clause) cout << "\t [ " << clause << " ] \n " ;
    } while (*clause) ;
}
```

```
Once upon a midnight dreary, while I pondered, weak and weary ,
Over many a quaint and curious volume of forgotten lore ,
^D [Once upon a midnight dreary]
    [while I pondered]
    [ weak and weary]
    [
Over many a quaint and curious volume of forgotten lore]
    [
]
```

لاحظ أن علامة نهاية السطر الغير مرئية التي تتلو "weary" مخزنة كالحرف الأول في سطر الدخل التالي . حيث أن الفاصلة مستخدمة كحرف الانهاء فإن حرف نهاية السطر يعالج كما لو كان حرفاً عادياً.

الدالة () cin. get تستخدم لقراءة حرف بحرف . والنداء cin. get (ch) تنسخ الحرف التالي من سيل الدخل cin في المتغير ch وتعيد 1 . إلا اذا اكتشفت نهاية الملف . في هذه الحالة تعيد 0 .

مثال 6.7 الدالة cin. get ()

يحسب هـذا البرنامـج عدد مرات حدوث الحرف ' e ' في سيل الدخل. تستمر الحلقة طالما كانت الدالة cin. get (ch) ناجحة في قراءة الحروف في المتغير ch :

```
main ()
{
    char ch.
    int count = 0 ;
    while (cin. get (ch) )
        if (ch == ' e' ) ++ count ;
    cout << count << " e ' s were counted. \n" ;
{
```

```
Once upon a midnight dreary, while I ponered, weak and weary,
Over many a quaint and curious volume of forgotten lore ,
^D
11 e' s were counted.
```

عكس get هو put . الدالة () cout. put تستخدم للكتابة في سيل الخرج cout حرف بحرف. يوضح ذلك بالمثال التالي:

مثال 7.7 الدالة cout. put ()

يردد هذا البرنامج سيل الدخل مع كتابة الحرف الأول لكل كلمة على الصورة المكبرة

```
#include < ctype.h >
main ()
{
    char ch, pre = '\0' ;
    while (cin. get (ch) ) {
        if (pre == ' ' || pre == '\n') cout. put (char (toupper (ch) ) ) ;
        else cout. put (ch) ;
        pre = ch ;
    }
}
```

ويكون الخرج على الصورة التالية

```
Fourscore and seven years ago our fathers
Fourscore And Seven Years Ago Our Fathers
brought forth upon this continent a new nation ,
Brought Forth Upon This Continent A New Nation ,
^D
```

المتغير pre يحتفظ بالحرف المقروء سابقاً . الفكرة أنه إذا كانت pre هي حرف خالي أو حرف سطر جديد فإن الحرف التالي ch سيكون الحرف الأول في الكلمة الجديدة . في هذه الحالة سيتبدل بنفس الحرف ولكن مكبراً 'ch + 'A' - 'a .

ملف الرأس <ctype.h> يعلن عن الدالة (toupper (ch والتي تعيد الحرف المكبر المكافئ لـ ch إذا كان ch هو حرف مصغر .

الدالة () cin. putback تعيد الحرف المقروء الأخير بواسطة () cin. get ثانية لسيل الدخل cin . الدالة () cin. ignore تقرأ حرف أو أكثر في سيل الدخل cin بدون معالجتهم . المثال 8.7 يوضح هذه الدوال.

الدالة () cin. peek يمكن استخدامها بدلاً من تجميع الدالتين () cin. get و cin. putback النداء:

```
ch = cin . peek ()
```

ينسخ الحرف التالي من سيل الدخل cin في المتغير ch من النوع char بدون ازالة هذا الحرف من سيل الدخل. المثال 9.7 يوضح كيف أن الدالة () peek يمكن استخدامها بدلاً من الدالتين () get و () putback .

مثال 8.7 الدوال () cin. putback و cin. ignore

هذا البرنامج يختبر دالة تستخلص الأعداد الصحيحة من سيل الدخل

```
int nextInt () ;

main ()
{
    int m = nextInt () , n = nextInt () ;
    cin. ignore (80, '\n') ;         // ignore rest of input line
    cout << m << " + " << n << " = " << m+n << endl ;
}

int nextInt ()
{
    char ch ;
    int n ;
    while (cin . get (ch) )
        if (ch >= '0' && ch <= '9') {  // next character is a digit
            cin . putback (ch) ;       // put it back so it can be
            cin >> n ;                 // read as a complete int
            break ;
        }
    return n ;
}
```

```
What is 305 plus 9416?
305 + 9416 = 9721
```

تفحص الدالة () nextInt الحروف في cin حتى تقابل أول رقم . في هذا التنفيذ الرقم هو 3 . حيث أن هذا الرقم سيكون أول جزء من الرقم الصحيح 305 ، فإنه يوضع ثانية في cin حتى يمكن قراءة العدد الصحيح كاملاً في n ثم يعاد.

مثال 9.7 الدالة () cin . peek

هذا الاصدار للدالة () nextInt يكافئ التي استخدمناها بالمثال السابق .

```
int nextInt ()
{
    char ch ;
    int n ;
    while (ch = cin . peek ( ) )
        if (ch >= '0' && ch <= '9' ) {
            cin >> n ;
            break ;
        }
        else cin.get (ch) ) ;
    return n;
}
```

التعبير () ch = cin.peek ينسخ الحرف التالي في ch ويعيد 1 إذا نجحت. عندئذ إذا كانت ch رقم فإن العدد الصحيح الكامل يُقرأ في n ويعاد . وإلا فإن الحرف يزال من cin وتستمر الحلقة . إذا قابلنا نهاية الملف فإن التعبير () ch = cin.peek يعيد 0 ، الذي يوقف الحلقة .

6.7 دوال الحروف المعرفة في ملف الراس <ctype.h>

المثال 7.7 يوضح الدالة () toupper أنها احدى مجموعة دوال معالجة الحروف والمعلنة في ملف الرأس <ctype.h> . تم تلخيص هذه الدوال بالجدول 1.7.

الجدول 1.7 الدوال <ctype.h>

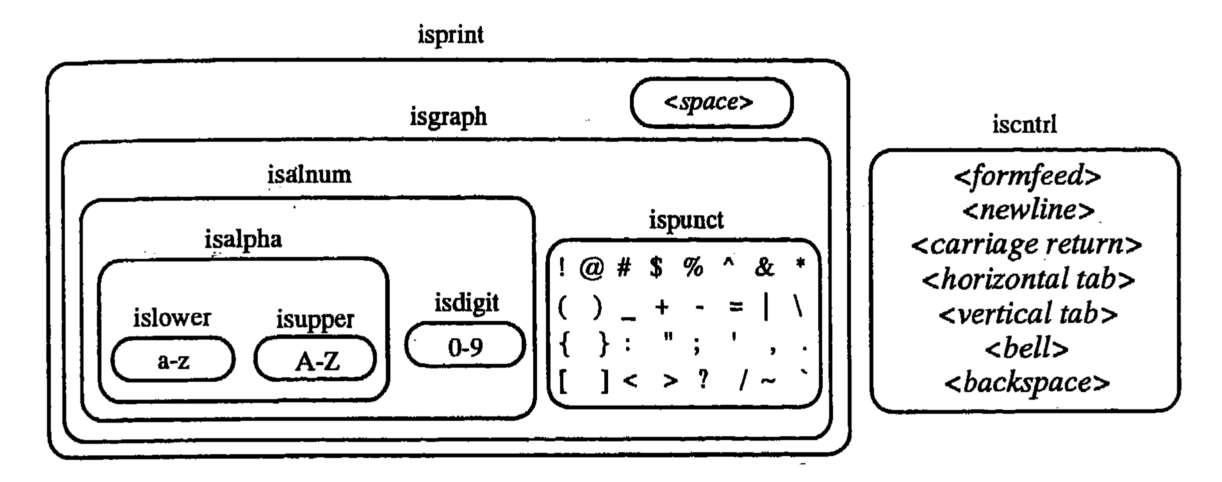
isalnum ()	int isalnum (int c) ; يعيد قيمة غير صفرية إذا كانت c حرف ابجدي أو رقم و إلا يعيد صفر
isalpha ()	int isalpha (int c) ; يعيد قيمة غير صفرية إذا كانت c حرف ابجدي وإلا يعيد صفر
iscntrl ()	int iscntrl (int c) ; يعيد قيمة غير صفرية إذا كانت c حرف تحكم وإلا يعيد صفر .
isdigit ()	int isdigit (int c) ; يعيد قيمة غير صفرية إذا كانت c هو حرف عددي وإلا فيعيد صفر.

تابع الجدول 1.7 الدوال <ctype.h>

isgraph ()	int isgraph (int c); يعيد قيمة غير الصفر إذا كانت c حرف مطبوع غير فاضي وإلا يعيد صفر.
islower ()	int islower (int c); يعيد قيمة غير الصفر إذا كانت c حرف ابجدي صغير وإلا فسيعيد صفر.
isprint ()	int isprint (int c); يعيد قيمة غير الصفر إذا كانت c أي حرف يطبع وإلا فسيعيد صفر.
ispunct ()	int ispunct (int c); يعيد قيمة غير الصفر إذا كانت c أي حرف مطبوع فيما عدا الحروف الابجدية. والحروف العددية والمسافة وإلا فسيعيد صفر.
isspace ()	int isspace (int c); يعيد قيمة غير الصفر إذا كانت c هي أي مسافة بيضاء ' ' . يقدم النموذج '\f' ، سطر جديد '\n' ، عودة العربة (ادخال) '\r' ، الجدولة (أو المجال) الافقية '\t' والجدولة الرأسية '\v' وإلا سيعيد صفر .
isupper ()	int isupper (int c); تعيد قيمة غير الصفر إذا كانت c هي حرف مكبر وإلا ستعيد صفر.
isxdigit ()	int isxdigit (int c); تعيد قيمة غير الصفر إذا كانت c احدى الأرقام العشرة أو احدى الارقام الستعشرية الاثنى عشرA+F ، a+f وإلا تعيد صفر.
tolower ()	int tolower (int c); تعيد الحرف مصغراً إذا كانت c حرف أبجدي مكبر وإلا فتعيد 0 .
toupper ()	int toupper (int c); تعيد الحرف مكبراً إذا كانت c حرف ابجدي مصغراً وإلا فستعيد صفراً.

لاحظ أن هذه الدوال تستقبل ثابت c صحيح int وتعيد int . يعمل هذا لأن char من النوع int . عادة يمرر الحرف char للدالة والقيمة المعادة تنسب إلى char ، ولهذا ننظر إليها كدوال تعديل الحروف.

هاهو التخطيط الذي يحمل غالبية دوال <ctype.h>.



إنها تبين على سبيل المثال إذا كان ch هو الحرف '$' فإن isprint (ch) و isgraph (ch) و ispunct (ch) ستعيد قيمة غير صفرية (أي "true") بينما isalnum (ch) و isalpha (ch) و islower (ch) ستعيد صفر (أي "false") .

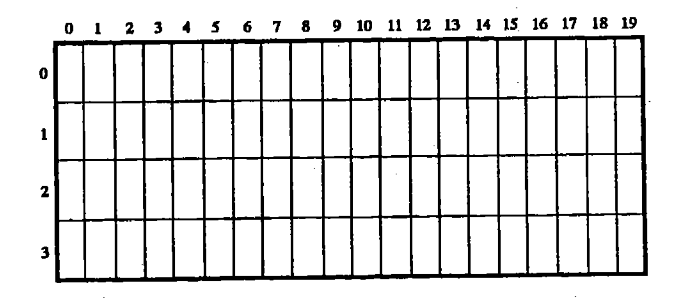
7.7 مصفوفات السلاسل

تذكر أن المصفوفات ثنائية الأبعاد في الحقيقة هي مصفوفة أحادية البعد عناصرها نفسها هي مصفوفة أحادية البعد. وعندما تكون عناصر هذه المصفوفة هي سلاسل . فإنه يكون لدينا مصفوفة سلاسل.

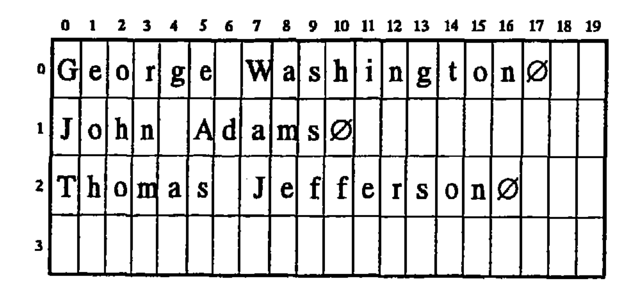
مثال 10.7 يعلن عن المصفوفة ثنائية الابعاد name كالآتي:

```
char name [4] [20] ;
```

يخصص هذا الإعلان 80 بايت منظمة كالتالي :



كل من الصفوف الأربع هو مصفوفة أحادية البعد مكونة من 20 حرف ولهذا ينظر إليها كسلسلة حروف، نصل إلى هذه السلاسل هكذا name [0] ، name [1] ، name [2] ، name [3] . في تنفيذ البرنامج الموضح بالمثال 10.7 تخزن البيانات كالتالي :



	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19
0	G	e	o	r	g	e		W	a	s	h	i	n	g	t	o	n	Ø		
1	J	o	h	n		A	d	a	m	s	Ø									
2	T	h	o	m	a	s		J	e	f	f	e	r	s	o	n	Ø			
3																				

هنا الرمز Ø يمثل "NUL" الحرف الصفري '\0' .

مثال 10.7 مصفوفة من السلاسل

هذا البرنامج يقرأ تتابع من السلاسل ثم يخزنها في مصفوفة ثم يطبعها

```
main ()
{
    char name  [8] [24] ;
    int count  =  0 ;
    cout  <<  "Enter at most  8  names with at most  23  characters: \n" ;
    while  (cin.getline  (name  [count++] ,  24))
        ;
    --count ;
    cout  <<  "The names are :  \n" ;
    for  (int  i  =  0 ;  i  <  count ;  i++)
        cout  <<  "\t"  <<  i  <<  " .  ["  <<  name  [i]  <<  "] "  <<  endl ;
}
```

```
Enter at most 8 names with at most 23 characters:
George Washington
John Adams
Thomas Jefferson
^D
The names are :
        0.  [George Washington]
        1.  [John Adams]
        2.  [Thomas Jefferson]
```

لاحظ أن كل النشاط في الحلقة while يتم خلال شرط التحكم

```
cin . getline  (name  [count ++] ,  20)
```

هذا النداء للدالة () cin . getline يقرأ السطر التالي في [name [count ثم يزيد count . تعيد الدالة قيمة غير صفرية (أي "true") إذا نجحت في قراءة سلسلة الحروف في [name [count . عند إشارة نهاية الملف (مع control-D أو control-z) تفشل الدالة () cin . getline ولهذا تعيد 0 التي توقف الحلقة while . جسم هذه الحلقة فارغاً والمشار إليها بالسطر الذي يحتوى على لا شيء سوى الفاصلة المنقوطة " ; " .

طريقة أكثر كفاءة لتخزين السلاسل هو أن نعلن عن مصفوفة مؤشرات :

```
char*  name  [4] ;
```

نجد أن كل من العناصر الأربع له النوع *char بمعنى أن كل [name [i هو سلسلة. في البداية لا يخصص هذا الاعلان أي تخزين لبيانات السلسلة . بدلاً من ذلك نحتاج إلى تخزين كل البيانات في سلسلة عازلة . عندها يمكن أن نضبط كل [name [i مساوياً لعنوان الحرف الأول للاسم المناظر في العازل كما تم تنفيذه في المثال 11.7 . هذه الطريقة أكفأ لأن كل عنصر من [name [i تستخدم فقط عدد من البايتات التي تحتاج اليها لتخزين السلسلة (بالاضافة لتخزين مؤشر واحد) . الثمن لذلك هو أن برنامج الدخل يحتاج إلى علامة تشير إلى نهاية الدخل.

مثال 11.7 مصفوفة سلسلة

يوضح هذا البرنامج استخدام الدالة () getline بالحرف '$' . إنها تقريباً تكافئ الطريقة المستخدمة في المثال 10.7 حيث يقرأ تتابع من الأسماء واحد لكل سطر منتهي بالعلامة السنتية '$' ثم تطبع الأسماء المخزنة في المصفوفة name :

```
main  ()
{
     char buffer  [80] ;
     cin . getline (buffer, 80, '$') ;
     char*  name  [4] ;
     name  [0]  =  buffer ;
     int count  =  0 ;
     for  (char*  p  =  buffer ;   *p  !=  '\0'  ;  p++)
          if   (*p  ==  '\n')   {
               *p  =  '\0' ;                        //  end name  [count]
               name  [++count]  =  p+1;         //  begin next name
          }
     cout  <<  "The names are :  \n" ;
     for  (int  i  =  0;   i  <  count;  i++)
          cout  <<  "\t"  <<  i  <<  "  .  ["  <<  name [i]  <<  "] "  <<  endl ;
}
```

George Washington

John Adams

Thomas Jefferson

$

The names are :

0. [George Washington]

1. [John Adams]

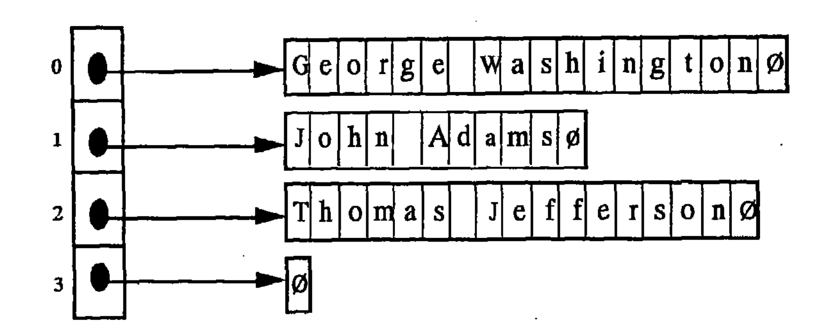
2. [Thomas Jefferson]The names are :

تم تخزين الدخل كاملاً في buffer كسلسلة واحدة تحتوي على :

"George Washington \ nJohn Adams \nThomas Jefferson \n"

تفحص الحلقة for خلال العازل باستخدام المؤشر p . كل مرة يجد المؤشر p الحرف '\n' ينهي السلسلة في name [count] بالحاق الحرف '\0' إليها . ثم تزيد العداد count وتخزن العنوان p + 1 للحرف التالي في name [count].

وتكون المصفوفة الناتجة مشابهة للآتي :



لاحظ أن البايتات الاضافية التي اضيفت في نهايات الاسماء في المثال 10.7 ليست مطلوبة. هنا إذا كانت السلاسل معلنة في وقت الترجمة فإن مصفوفة السلسلة المعرفة عالية هي أبسط بكثير للتداول. المثال 12.7 يوضح كيف يتم بدأ مصفوفة السلسلة .

مثال 12.7 بدأ مصفوفة السلسلة

هذا البرنامج يكافئ تقريباً ما جاء بالمثالين السابقين حيث يبدأ مصفوفة السلسلة name ثم تطبع محتوياتها.

```
main ()
{
    char* name [ ] =  { "George Washington" ,
                        "Johan Adams" ,
                        "Thomas Jefferson"
                      } ;
    cout << "The names are : \n" ;
    for (int i = 0 ;  i < 3 ;  i++)
        cout << "\t" << i << " . [ " << name [i] << " ] " << endl;
}
```

```
The names are :
        0.  [George Washington]
        1.  [John Adams]
        2.  [Thomas Jefferson]
```

التخزين للبيانات في المصفوفة name هنا هو نفسه كما بالمثال 11.7 .

8.7 مكتبة لغة C للتعامل مع السلسلة

ملف الرأس للغة C <string.h> أيضاً يسمى مكتبة سلسلة C. يتضمن مجموعة دوال هامة جداً للتعامل مع السلاسل . المثال 13.7 يوضح أبسط هذه الدوال. دالة طول السلسلة التي تعيد طول السلسلة التي ترسل لها.

مثال 13.7 دالة طول السلسلة () strlen

هذا هو برنامج بسيط لاختبار الدالة () strlen . النداء (s) strlen يعيد ببساطة عدد الحروف في s التي تسبق أول حدوث للحرف الصفري '\0' :

```
main ()
{
    char s [] = "ABCDEFG" ;
    cout << "strlen ( " << s << " ) = " << strlen (s) << endl ;
    cout << "strlen ( \" \" ) = " << strlen ( " " ) << endl ;
    char buffer [80] ;
    cout << "Enter string: "; cin >> buffer ;
    cout << "strlen ( " << buffer << " ) = " << strlen (buffer) << endl;
}
```

```
strlen (ABCDEFG) = 7
strlen (" ") = 0
Enter string: computer
strlen (computer) = 8
```

في بعض الطرق، تسلك السلاسل مثل الأهداف الأساسية (أي أعداد صحيحة وأرقام حقيقية) . على سبيل المثال يمكن إخراجها إلى cout بنفس الطريقة. ولكن السلاسل هي أهداف منشأة تتكون من أجزاء أصغر (حروف) . لهذا فإن كثير من العمليات المتاحة للأهداف الاساسية مثل عامل التنسيب (=) ، عوامل المقارنة (<= ، >= ، == ، < ، > ، =!) والعوامل الحسابية (+ ، - ، ... الخ) ليست متاحة للسلاسل. بعض الدوال في مكتبة C للسلاسل تحاكي هذه العمليات . في الفصل الثامن سنتعلم كيف نكتب هذه العمليات بطريقتنا .

المثال التالي يوضح ثلاث دوال أخرى للسلاسل . تستخدم هذه الدوال في تحديد حروف وسلاسل جزئية في سلسلة معطاة.

مثال 14.7 الدوال () strchr . () strrchr و () strstr

```
#include <string.h>
main ()
{
    char s [] = "The Mississippi is a long river. " ;
    cout << "s = \ " " << s << "\" \n" ;
    char* p = strchr (s, ' ') ;
    cout << "strchr (s, ' ') points to s [ " << p - s << " ] . \n " ;
    p = strchr (s, 's') ;
    cout << "strchr (s, 's' ) points to s [ " << p - s << " ] . \n " ;
    p = strrchr (s, 's') ;
    cout << "strrchr (s, 's' ) points to s [ " << p - s << " ] . \n " ;
    p = strstr (s, "is") ;
    cout << "strstr (s, \ "is" \) points to s [ " << p - s << " ] . \n " ;
    p = strstr (s, "isi") ;
    if (p == NULL) cout << "strstr (s, \"isi\") returns NULL\n" ;
}
```

ها هو الخرج :

```
s = "The Mississippi is a long river. "
strchr (s, ' ') points to s [3] .
strchr (s, 's') points to s [6] .
strrchr (s, 's') points to s [17] .
strstr (s, "is") points to s [5].
strstr (s, "isi") returns NULL
```

النداء (' ' ,strchr (s يعيد مؤشر لأول ورود للحرف الفارغ ' ' في السلسلة s . التعبير p-s يحسب الدليل (ازاحة) 3 لهذا الحرف بالسلسلة . (تذكر أن المصفوفات استخدمت دليل مرتكز على الصفر لذلك فحرف البداية 'T' له الدليل 0.) . بالمثل الحرف 's' يظهر أولاً عند الدليل 6 في s .

النداء (' ' ,strrchr (s يعيد مؤشر لآخر ورود للحرف 's' في السلسلة s ، هذا هو الحرف s [17].

النداء ('is' ,strstr (s يعيد مؤشر لأول ورود للسلسلة الجزئية 'is' في السلسلة s ; هذا عند s [5].

النداء ("isi" ,strstr (s تعيد المؤشر NULL لأن "isi" لم توجد في أي مكان بالسلسلة s.

يوجـــد دالتـــين يحاكـيان عامـــل التنســيب للسلاسل هما : () strcpy و () strncpy . النداء (strcpy (s1, s2 ينسخ السلسلة s2 في السلسلة s1 . والنداء (strncpy (s1, s2, n ينسخ أول n من حروف السلسلة s1 . كلا الدالتين يعيدان s1 . هاتان الدالتان موضحتان في المثالين التاليين .

مثال 15.7 دالة نسخ السلسلة () strcpy

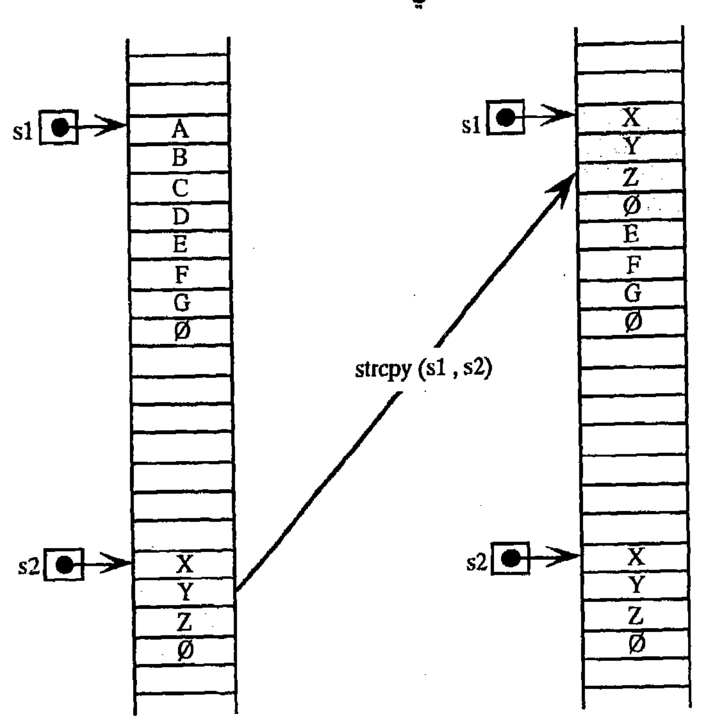
يتتبع هذا البرنامج النداء (strcpy (s1, s2 :

```
#include <iostream.h>
#include <string.h>
main ()
{
    char s1 [ ] = "ABCDEFG" ;
    char s2 [ ] = "XYZ" ;
    cout << "Before strcpy (s1, s2) : \n " ;
    cout << \ts1 = [ " << s1 << " ] , length = " << strlen (s1) << endl;
    cout << \ts2 = [ " << s2 << " ] , length = " << strlen (s2) << endl;
    strcpy (s1, s2) ;
    cout << "After strcpy (s1, s2) : \n ;
    cout << "\ts1 = [ " << s1 << " ] , length = " << strlen (s1) << endl;
    cout << "\ts2 = [ " << s2 << " ] , length = " << strlen (s2) << endl;
}
```

```
Before strcpy (s1, s2) :
        s1 = [ABCDEFG] , length = 7
        s2 = [XYZ] , length = 3
After strcpy (s1, s2) :
        s1 = [XYZ] , length = 3
        s2 = [XYZ] , length = 3
```

بعد نسخ s2 في s1 لا يمكن تمييزهما : كلاهما تتكون من ثلاث حروف XYZ .

تأثير (s1, s2) strcpy يمكن تصوره كالآتي :



حيث أن s2 لها الطول 3 و (s1, s2) strcpy ينسخ 4 بايتات (شاملة الحرف الصفري NUL ، موضح ф) تكتب على أول أربعة حروف من s1 ، يعدل هذا الأمر طول s1 إلى 3.

لاحظ أن (s1, s2) strcpy يكون نسخة مرادفة للسلسلة s2 . النسختان الناتجتان منفصلتان بحيث أن تغيير احداهما ليس له تأثير على الأخرى .

مثال 16.7 الدالة الثانية لنسخ السلسلة strncpy ()

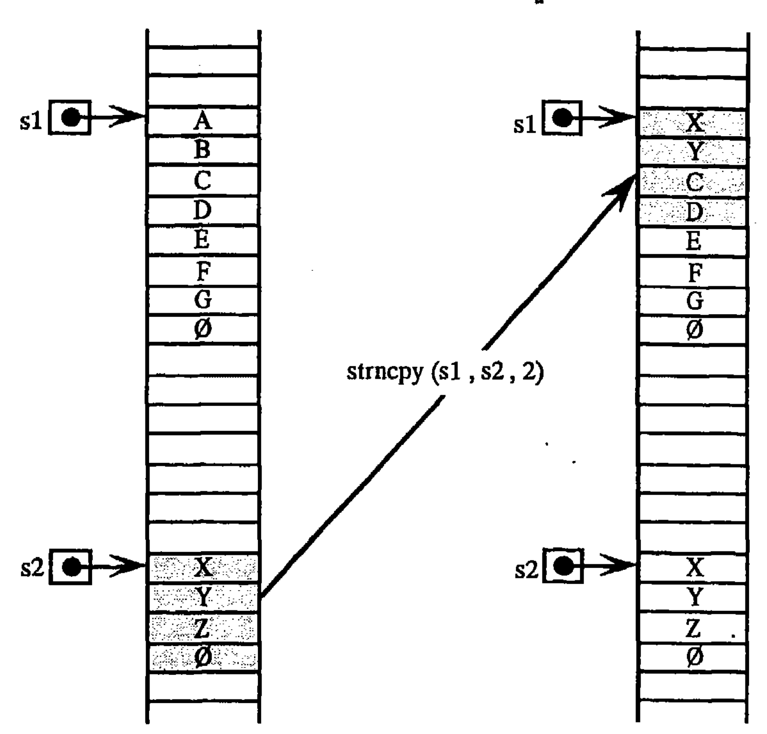
يتتبع هذا البرنامج النداءات (s1, s2, n) strncoy :

```
#include <iostream.h>
#include <string.h>
// Test-driver for the strncpy () function:
main ()
{
    char s1 [] = "ABCDEFG";
    char s2 [] = "XYZ";
    cout << "Before strncpy (s1, s2, 2) : \n";
    cout << "\ts1 = [" << s1 << "], length = " << strlen (s1) << endl;
    cout << "\ts2 = [" << s2 << "], length = " << strlen (s2) << endl;
```

```
    strncpy (s1, s2, 2);
    cout << "After strncpy (s1, s2, 2): \n";
    cout << "\ts1 = [" << s1 << "], length = " << strlen(s1) << endl;
    cout << "\ts2 = [" << s2 << "], length = " << strlen(s2) << endl;
}
```

```
Before strncpy (s1, s2, 2):
        s1 = [ABCDEFG], length = 7
        s2 = [XYZ], length = 3
After strncpy (s1, s2, 2):
        s1 = [XYCDEFG], length = 7
        s2 = [XYZ], length = 3
```

النداء (s1, s2, 2) strncpy يستبدل أول حرفين من s1 بـ XY ، تاركة بقية s1 بدون تغيير . تأثير strncpy (s1, s2, 2) يمكن تصوره كالتالي :



حيث أن طول s2 هو 3 ، (s1, s2, 2) strncpy ينسخ 2 بايت (باستثناء الحرف الصفري Ø : NUL) تكتب على أول حرفين في s1 . وليس له تأثير على طول s1 الذي هو 7.

إذا كانت n < strlen (s2) كما في المثال السابق إذن strncpy (s1, s2, n) ببساطة تنسخ أول n من حروف s2 في بداية s1 . ومع ذلك إذا كانت n ≥ strlen (s2) فإن strncpy (s1, s2, n) لها نفس التأثير مثل strcpy (s1, s2) : تجعل s1 نسخة من s2 بنفس الطول.

الدوال () strcat و () strncat تعمل مثل الدوال () strcpy و () strncpy فيما عدا أن الحروف من السلسلة الثانية تنسخ في نهاية السلسلة الأولى . المصطلح “cat” يأتي من الكلمة “catenate” تعني توصيل السلاسل معاً.

مثال 17.7 دالة توصيل السلسلة () strcat

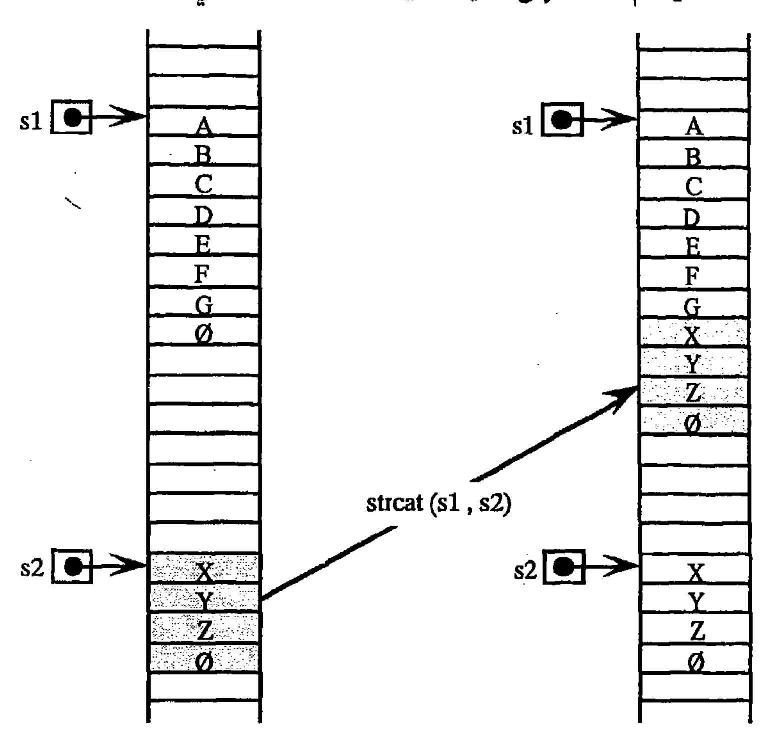
يتتبع هذا البرنامج نداء (strcat (s1, s2 التي تلحق s2 في نهاية السلسلة s1 :

```
#include <iostream.h>
#include <string.h>
// Test-driver for the strcat () function :
main ()
{
    char s1 [] = "ABCDEFG";
    char s2 [] = "XYZ";
    cout << "Before strcat (s1, s2) : \n";
    cout << "\ts1 = [" << s1 << "], length = " << strlen (s1) << endl;
    cout << "\ts2 = [" << s2 << "], length = " << strlen (s2) << endl;
    strcat (s1, s2);
    cout << "After strcat (s1, s2) : \n";
    cout << "\ts1 = [" << s1 << "], length = " << strlen (s1) << endl;
    cout << "\ts2 = [" << s2 << "], length = " << strlen (s2) << endl;
}
```

وها هو الخرج

```
Before strcat (s1, s2) :
        s1 = [ABCDEFG], length = 7
        s2 = [XYZ], length = 3
After strcat (s1, s2) :
        s1 = [ABCDEFGXYZ], length = 10
        s2 = [XYZ], length = 3
```

النداء (strcat (s1, s2 يضم XYZ إلى نهاية s1 يمكن تصورها كالآتي :



حيث أن طول s2 هو 3 ، تنسخ (strcat (s1, s2 4 بايت (شاملة الحرف الصفري NUL موضحة Ø) كاتبة على الحرف الصفري NUL للسلسلة s1 والبايتات الثلاث التالية . لقد زاد طول s1 إلى 10 .

إذا كانت أي من البايتات الزيادة التالية لـ s1 والمطلوب لنسخ s2 مستخدمة بواسطة أي هدف آخر فإن كل s1 والمنضم لها s2 ستنسخ إلى قسم آخر خالي بالذاكرة.

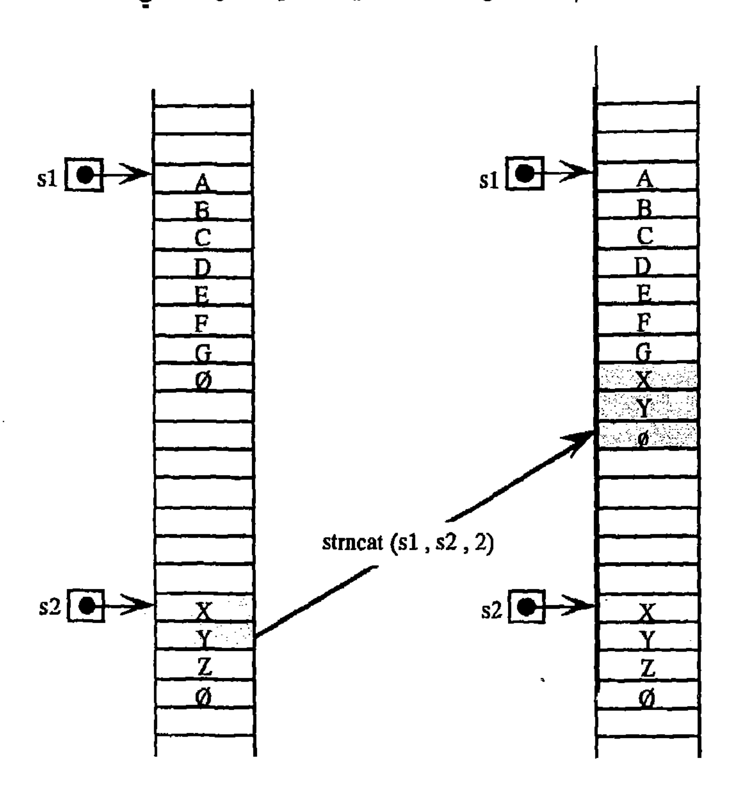
مثال 18.7 دالة الالحاق الثانية للسلسلة (strncat (s1, s2, n:

```
#include <iostream.h>
#include <string.h>
//  Test-driver for the strncat () function :
main ()
{
    char s1 [] = "ABCDEFG" ;
    char s2 [] = "XYZ" ;
    cout << "Before strncat (s1, s2, 2) : \n" ;
    cout << "\ts1 = [ " << s1 << " ], length = " << strlen (s1) << endl;
    cout << "\ts2 = [ " << s2 << " ], length = " << strlen (s2) << endl;
    strncat (s1, s2, 2) ;
    cout << "After strncat (s1, s2, 2) : \n" ;
    cout << "\ts1 = [ " << s1 << " ], length = " << strlen (s1) << endl;
    cout << "\ts2 = [ " << s2 << " ], length = " << strlen (s2) << endl;
}
```

الخرج يشبه الآتي :

Before strncat (s1, s2, 2) :
s1 = [ABCDEFG], length = 7
s2 = [XYZ], length = 3
After strncat (s1, s2, 2) :
s1 = [ABCDEFGXY], length = 9
s2 = [XYZ], length = 3

النداء strncat (s1, s2, 2) يضم XY على نهاية s1 . ويمكن رؤية التأثير كالآتي :



حيث أن طول s2 هو 3 بايتات فإن النداء strncat (s1, s2, 2) ينسخ 2 بايت وتكتب فوق الحرف الصفري NUL في s1 والبايت التالية له . ثم تضيف الحرف NUL في البايت التالية لتكتمل السلسلة s1 . ذلك يزيد طولها إلى 9 بايت (إذا كانت أي من الـ 2 بايت الزيادة مستخدمة بواسطة هدف آخر ، فإن كل الحروف العشرة ABCDEFGXYZ ϕ من الممكن أن تكتب في جزء آخر خالي من الذاكرة) .

يوضح المثال التالي دالة تقطيع السلسلة String tokenize function الغرض منها التعرف على مقاطع من سلسلة معطاة . أي كلمات في عبارة .

مثال 19.7 دالة تقطيع السلسلة () strtok

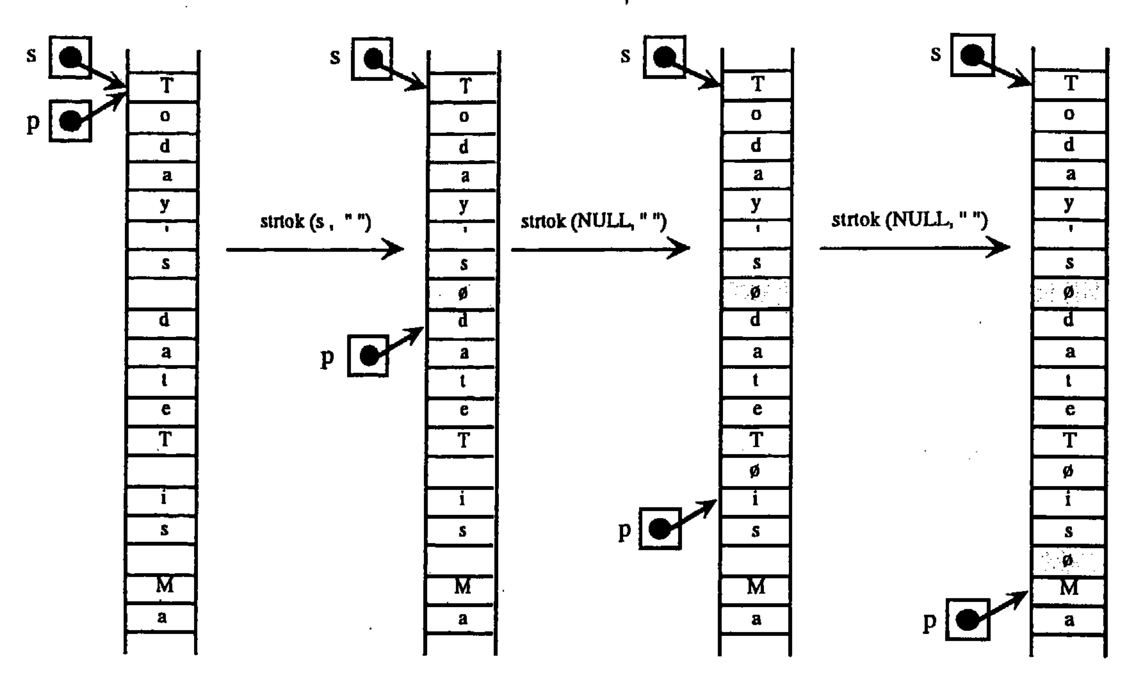
هذا البرنامج يبين كيف تستخدم الدالة () strtok لاستخلاص الكلمات من جملة .

```
#include  <iostream.h>
#include  <string.h>
//  Test-driver for the strtok () function :
main ()
{
     char  s [] = "Today's date is March 12,  1995. " ;
     char*  p ;
     cout << "The string is :  [" << s << "] \nIts tokens are : \n" ;
     p = strtok (s, "  ") ;
     while (p)  {
          cout << " \t [ " << p << " ] \n " ;
          p = strtok (NULL, "   ") ;
     }
     cout << "Now the string is :  [ " << s << " ] \n" ;
}
```

```
The string is :  [Today's date is March 12 , 1995. ]
Its tokens are :
               [Today's]
               [date]
               [is]
               [March]
               [12 ,]
               [1995.]
Now the string is :  [Today's]
```

النداء (" " ,p = strtok (s يحدد المؤشر p ليشير إلى الجزئية الأولى بالسلسلة s ويغير المسافة الخالية التي تلي "Today's" إلى الحرف الصفري NUL ' 0\ ' (المشار إليه بالرمز φ في التخطيط التالي) . تأثير ذلك هو أن أصبحت كل من s ، p هي السلسلة "Today's". بعدها كل نداء تالي (" " ,p = strtok (NULL يقدم المؤشر إلى حرف غير خالي تالي للحرف الصفري الجديد NUL . تغيير كل مسافة خالية تمر عليها إلى حرف صفري NUL . وتغيير أول حرف تالي p* إلى الحرف الصفري NUL . ذلك له تأثير جعل p كسلسلة جزئية تالية للتي كانت تنتهي بمسافات والآن تنتهي بالحرف الصفري NUL . يستمر ذلك حتى تصل p إلى الحرف NUL الذي ينهي السلسلة الأصلية s . هذا يجعل p = (NULL) أي الصفر ويوقف حلقة while . التأثير المجمل على السلسلة الأصلية s نتيجة كل النداءات لـ () strtok هو تغيير

كل مسافة خالية إلى NUL . هذا يجزئ السلسلة s . ويغيرها إلى تتابع من سلاسل جزئية محددة يكون الأول فقط منها معرفة بـ s . لاحظ أن الدالة () strtok تغير السلسلة أي تجزئها لهذا إذا أردت استخدام السلسلة الأصلية بعد تجزئتها فلابد أن تنسخها باستخدام () strcpy.



لاحظ أيضاً أن المعامل الثاني لدالة () strtok هو سلسلة . تستعمل هذه الدالة في هذه السلسلة كمنهيات في السلسلة الأولى . كمثال للتعرف على الكلمات في s ، يمكن استخدام (" . ; : , " ,strtok (s.

الدالة () strpbrk أيضاً تستخدم سلسلة الحروف كتجميع للحروف . إنها تعمم الدالة () strchr ، بالنظر لأول حدوث في السلسلة الأولي على أي من الحروف في السلسلة الثانية.

مثال 20.7 الدالة () strpbrk

```
#include <iostream.h>
#include <string.h>
// Test-driver for the strtok () function:
main ()
{
    char s [] = "The Mississippi is a long river. " ;
    cout << "s = \" " << s << " \" n " ;
    char* p = "strpbrk (s, "nopqr") ;
    cout << "strpbrk (s, \ "nopqr \ ") points to s [ " << p - s << " ]. \n";
    p = strbprk (s, "NOPQR" ) ;
    if (p == NULL) cout << "strpbrk (s, \ "NOPQR\ ") returns NULL. \n" ;
}
```

```
s = "The Mississippi is a long river. "
strpbrk (s,    "nopqr" ) points to s [12] .
strpbrk (s,    "NOPQR") returns NULL.
```

النداء ("strpbrk (s,   "nopqr يعيد أول ورود في s لأي من الحروف الخمسة 'n' , 'o' , 'p' , 'q' , أو 'r'. الحرف الأول الموجود هو 'p' عند [12] s .

النداء ("strpbrk (s,   "NOPQR يعيد المؤشر الصفري (NUL) لأنه لا يوجد في s أي من هذه الحروف الخمسة .

الجدول 2.7 يجمل بعض الدوال الهامة المعلنة في <string.h> . لاحظ أن نوع size-t هو عدد صحيح معرف بالملف <string.h> .

الجدول 2.7 الدوال <string.h>

memcpy ()	void* memcpy (void* s1, const void* s2, size-t n) ; تستبدل عدد n بايت الأولى من s1* بعدد الـ n بايت الأولى من s2* . تعيد s.
strcat ()	char* strcat (char* s1, const char* s2) ; تلحق s2 إلى s1 وتعيد s1.
strchr ()	char* strchr (const char* s , int c) ; تعيد أول حدوث للحرف c في s وتعيد NULL إذا لم يكن c ضمن s.
strcmp ()	int strcmp (const char* s1, const char* s2) ; تقارن s1 بالسلسلة الجزئية s2 . تعيد رقم صحيح سالب أو صفر أو رقم صحيح موجب بناءاً على ما إذا كان s1 ترتيباً أقل أو يساوي أو أكبر من s2.
strcpy ()	char* strcpy (char* s1, const char* s2) ; تستبدل s1 بـ s2 وتعيد s1 .
strcspn ()	size-t strcspn (char* s1 , const char* s2) ; يعيد طول أطول سلسلة فرعية في s1 والتي تبدأ بـ [0] s1 . ولا تحتوي على أي حرف موجود في s2 .
strlen ()	size-t strlen (const char* s) ; تعيد طول s الذي هو عدد الحروف اعتباراً من [0] s التي تسبق أول وجود للحرف الصفري NUL.

تابع الجدول 2.7 الدوال <string.h>

strncat ()	char* strncat (char* s1 , const char* s2, size-t n) ; تلحق أول n من حروف s2 إلى s1 ثم تعيد s1 . إذا كانت n ≥ strlen (s2) فإن strncat (s1, s2, n) لها نفس التأثير مثل strcat (s1, s2) .
strncmp ()	int strncmp (const char* s1, const char* s2, size-t n); تقارن أول n حرف في السلسلة s1 مع أول n حرف في السلسلة s2. تعيد رقم سالب أو صفر ، أو موجب بناءاً على ما إذا كانت s1 حرفياً أقل ، تساوي أو أكبر من s2 ، إذا كانت n ≥ strlen (s2) فإن strncmp (s1, s2, n) ، strcmp (s1, s2) لهما نفس التأثير.
strncpy ()	char* strncpy (char* s1, const char* s2, size-t n) ; تستبدل أول n حروف من s1 بأول n حروف من s2 وتعيد s1. إذا كانت n ≤ strlen (s1) فإن طول s1 لا يتأثر . إذا كانت n ≥ strlen (s2) فإن strncpy (s1, s2, n) ، strcpy (s1, s2) لهما نفس التأثير.
strpbrk ()	char* strpbrk (const char* (s1, const char* s2) ; يعيد عنوان أول حدوث في s1 لأي من الحروف في s2 . يعيد الحرف الصفري (NULL) إذا لم يظهر أي من حروف s2 في s1.
strrchr ()	char* strrchr (const char* s, int c) ; يعيد مؤشر لآخر وجود للحرف c في s . يعيد NULL إذا لم توجد c في s .
strspn ()	size-t strspn (char* s1, const char* s2) ; تعيد طول أطول سلسلة جزئية في s1 التي تبدأ بـ s1 [0] وتحتوي فقط على الحروف الموجودة في s2 .
strstr ()	char* strstr (const char* s1, const char* s2) ; يعيد عنوان أول حدوث للسلسلة s2 كسلسلة فرعية من s1 يعيد NULL إذا كانت s2 ليست في s1 .
strtok ()	char* strtok (char* s1, const char* s2) ; تجزئ السلسلة s1 إلى مقاطع بالحروف الموجودة في s2 بعد النداء الأولي strtok (s1, s2). كل نداء تالي strtok (NULL, s2) يعيد مؤشر إلى المقطع التالي الذي وجد في s1 . تغير هذه النداءات السلسلة s1 . تستبدل كل ناهي بالحرف الصفري '\0' .

أسئلة مراجعة

1.7 اعتبر الاعلانات التالية للسلسلة s :

```
char   s [6] ;
char   s [6]  =  { 'H', 'e', 'l', 'l', 'o' } ;
char   s [6]  =  "Hello" ;
char   s [ ] ;
char   s [ ]  =  new char [6] ;
char   s [ ]  =  { 'H', 'e', 'l', 'l', 'o' } ;
char   s [ ]  =  "Hello" ;
char   s [ ]  =  new  ("Hello") ;
char*   s ;
char*   s  =  new char [6] ;
char*   s  =  { 'H', 'e', 'l', 'l' , 'o' } ;
char*   s  =  "Hello" ;
char*   s  =  new ( "Hello" ) ;
```

أ – أي من هذه يمثل اعلان لسلسلة طبقاً للغة البرمجة C++ ؟

ب – أي من هذه يمثل اعـلان لسلسلة حـروف في C++ طولها 5 حروف تبدأ بالسلسلة "Hello" ومخصصة في وقت الترجمة؟

جـ – أي من هذه يمثل اعلان في C++ لسلسلة حروف طولها 5 ، تبدأ بالسلسلة "Hello" ومخصصة في وقت التنفيذ ؟

د – أي من هذه يمثل اعلان في C++ لسلسلة حروف كمعامل اساسي لدالة ؟

2.7 ما هو الخطأ في استخدام الجملة

```
cin >> s ;
```

لقراءة الدخل " Hello , World " في السلسلة s ؟

3.7 ما الذي تطبعه الشفرة التالية ؟

```
char s[ ] = "123 w.  42nd st.,  NY, NY 10020-1095";
int  count = 0;
for (char* p = s;  *p; p++)
     if (isupper (*p)) ++ count;
cout << count << endl;
```

4.7 ماذا تطبع الشفرة التالية ؟

```
char s[] = " 123 w. 42nd st., NY, NY  10020-1095";
for (char* p = s;  *p; p++)
     if (isupper (*p) )  *p = tolower (*p);
cout << s << endl;
```

5.7 ماذا تطبع الشفرة التالية :

```
char s[ ] = "123 W. 42nd st., NY, Ny 10020-1095 ";
for (char* p = s;  *p;  p++)
     if (isupper (*p) )  (*p) ++;
cout << s << endl;
```

6.7 ماذا تطبع الشفرة التالية :

```
char s[] = "123  W. 42nd st., NY, NY  10020-1095";
int count = 0;
for (char* p = s; *p; p++)
     if (ispunct  (*p) ) ++ count;
cout << count << endl;
```

7.7 ماذا تطبع الشفرة التالية :

```
char s[ ] = " 123 W. 42nd st., NY, NY 10020-1095";
for (char* p = s; *p; p++)
if (ispunct (*p) )  *(p-1) = tolower (*p);
cout << s << endl;
```

8.7 ما الفرق بين الأمرين التاليين إذا كانت s1 ، s2 لهما النوع char*

```
char* :
     s1 = s2;
     strcpy (s1, s2);
```

9.7 اذا كانت first تحتوي على السلسلة "Rutherford" ، last تحتوي على السلسلة "Hayes" فماذا يكون تأثير كل من النداءات التالية :

```
a.  int n = strlen (first) ;
b.  char* s1 = strchr (first, 'r') ;
c.  char* s1 = strrchr (first, 'r') ;
d.  char* s1 = strpbrk (first, "rstuv" ) ;
e.  strcpy (first, last) ;
f.  strncpy (first, last, 3) ;
g.  strcat (first, last) ;
h.  strncat (first, last, 3) ;
```

10.7 ماذا ينسب كل مما يأتي إلى n :

```
a.  int n = strspn ( "abecedarian" , "abcde" ) ;
b.  int n = strspn ( "beefeater" , "abcdef" ) ;
c.  int n = strspn ( "baccalaureate" , "abc" ) ;
d.  int n = strcspn ( "baccalaureate" , "rstuv" ) ;
```

11.7 ماذا تطبع الشفرة التالية :

```
char* s1 = "ABCDE" ;
char* s2 = "ABC" ;
if (strcmp (s1, s2) < 0)  cout << s1 << " < " << s2 << endl ;
else cout << s1 << " >= "  << s2 << endl;
```

12.7 ماذا تطبع الشفرة التالية :

```
char* s1 = "ABCDE" ;
char* s2 = "ABCE" ;
if (strcmp (s1, s2) < 0) cout << s1 << " < " << s2 << endl;
else cout << s1 << " >= " << s2 << endl ;
```

13.7 ماذا تطبع الشفرة التالية :

```
char* s1 = "ABCDE";
char* s2 = " ";
if (strcmp (s1, s2) < 0) cout << s1 << " < " << s2 << endl;
else cout << s1 << " >= " << s2 << endl;
```

14.7 ماذا تطبع الشفرة التالية :

```
char* s1 = " ";
char* s2 = " ";
if (strcmp (s1, s2) == 0) cout << s1 << " == " << s2 << endl;
else cout << s1 << " != " << s2 << endl;
```

مسائل محلولة

15.7 اشرح السبب في عدم استعمال البديل التالي للمثال 12.7

```
main ()
{
    char name [10] [20] , buffer [20];
    int count = 0;
    while (cin . getline (buffer, 20) )
        name [count] = buffer;
    --count;
    cout << "The names are : \n";
    for (int i = 0; i < count; i++)
        cout << "\t" << i << ". [ " << name [i] << " ] " << endl;
}
```

هذه الشفرة لا تعمل لأن أمر التنسيب

```
name [count] = buffer;
```

ينسب نفس المؤشر لكل من السلاسل name [0] ، name [1] ، إلخ.

المصفوفات لا يمكن أن تنتسب بهذه الطريقة . لنسخ مصفوفة في أخرى استخدم الدالة strcpy () أو الدالة strncpy () .

مسائل برمجة محلولة

16.7 اكتب الدالة () strcpy

لكي تنسخ السلسلة s2 في السلسلة s1 :

```
char* strcpy (char* s1 , const char* s2)
{
    for (char* p = s1 ; *s2 ; )
        *p++ = *s2++ ;
    *p = '\0' ;
    return s1 ;
}
```

يبدأ المؤشر p عند بداية s1 . في كل دورة من الحلقة for يتم نسخ الحرف s2* في الحرف p* عندئذ تزاد قيمة كلا من s2 و p . تستمر الحلقة حتى تصبح NULL = *s2 (أي الحرف الصفري '0\') . عند ذلك يضم الحرف الصفري للسلسلة s1 بتنسيبه إلى p* (لقد ترك المؤشر p مشيراً إلى البايت التالية لآخر بايت تم نسخها عند انتهاء الحلقة) .

لاحظ أن هذه الدالة لا تخصص أي تخزين جديد . لذلك فإن أول معامل لها s1 لابد أن تعرف مسبقاً على أنها سلسلة حروف بنفس الطول مثل s2 .

17.7 اكتب الدالة () strncat :

تضيف هذه الدالة حتى n من الحروف من s2 في نهاية s1 . هي نفسها كالدالة () strcat فيما عدا أن معاملها الثالث n يحدد عدد الحروف المنسوخة :

```
char* strncat (char* s1, const char* s2, size_t n)
{
    for (char* end = s1 ; *end ; end ++)          // find end of s1
    for (char* p = s2 ;  *p && p-s2 < n ; )
        *end++ = *p++ ;
    *end = '\0' ;
    return s1 ;
}
```

أول حلقة for تجد نهاية السلسلة s1 ، حيث عندها تضيف الحروف من السلسلة s2 . ثاني حلقة for تنسخ الحروف من s2 في الاماكن التالية لـ s1 . لاحظ كيف يعمل الشرط الاضافي n < s2 - q على الحد من الحروف المنسوخة إلى n : التعبير q - s2 يساوي عدد الحروف المنسوخة لأنها الفرق بين q (التي تشير إلى الحرف التالي المطلوب نسخة) و s2 (التي تشير إلى بداية السلسلة) لاحظ أن هذه الدالة لا تخصص أي تخزين جديد . إنها تتطلب أن السلسلة s1 يحتوي على الأقل على k بايت اضافية مخصصة ، حيث k هي الأصغر بين كلا من n وطول السلسلة s2 .

18.7 اكتب واختبر دالة تعيد صيغة الجمع plural للكلمة الانجليزية المرسلة اليها. يتطلب هذا اختبار الحرف الاخير والذي قبله بالكلمة المراد جمعها . نستعمل المؤشرات p و q للوصول لتلك الحروف .

```
void pluralize (char* s)
{
    int len = strlen (s);
    char* p = s + len - 1;          // last letter
    char* q = s + len - 2;          // last 2 letters
    if (*p == 'h' && (*q == 'c' || *q == 's')) strcat (p, " s ");
    else if (*p == 's') strcat (p, "es");
    else if (*p == 'y')
        if (isvowel (*q)) strcat (p, "s");
        else strcpy (p, "ies");
    else if (*p == 'z')
        if (isvowel (*q)) strcat (p, "zes");
        else strcat (p, "es");
    else strcat (p, "s");
}
```

اثنان من الاختبارات تعتمد على ما إذا كان الحرف الثاني من الأخير هو متحرك vowel ولهذا نعرف دالة منطقية صغيرة (بولينية) () isvowel لاختبار هذا الشرط :

```
int isvowel (char c)
{
    return (c == 'a' || c == 'e' || c == 'i' || c == 'o' || c == 'u');
}
```

برنامج الاختبار يكرر قراءة الكلمات ، يطبعها ، يجمعها ، يطبعها ثانية . تنتهي الحلقة عندما يدخل المستخدم مسافة واحدة خالية كما لو كانت كلمة .

```
#include <iostream.h>
#include <string.h>

void pluralize (char*) ;

main ()
{
     char word [80] ;
     for (; ;) {
          cin . getline (word, 80) ;
          if (*word == ' ') break ;
          cout << " \tThe singular is [ " << word << " ] . \n" ;
          pluralize (word) ;
          cout << " \t The plural is [ " << word << " ] . \n" ;
     }
}
```

```
wish
     The singular is [wish] .
     The plural is [wishes] .
hookah
     The singular is [hookah] .
     The plural is [hookahs] .
bus
     The singular is [bus] .
     The plural is [buses] .
toy
     The singular is [toy] .
     The plural is [toys] .
navy
     The singular is [navy] .
     The plural is [navies] .
quiz
     The singular is [quiz] .
     The plural is [quizzes] .
quartz
     The singular is [quartz] .
     The plural is [quartzes] .
computer
     The singular is [computer] .
     The plural is [computers] .
```

19.7 اكتب برنامج ليقرأ اسماء متتابعة ، اسم لكل سطر ثم يرتبهم ويطبعهم .

من المفترض أن الاسماء لا تحتوي على أكثر من 25 حرف ولا يزيد العدد عن 25 اسم. سنقرأ الدخل مرة واحدة وتخزنها في عازل واحد . حيث أن كل اسم سينتهي بحرف صفري NUL فيجب أن يكون العازل كبيراً بما يكفي لعدد [25 x (1+20) + 1] من الحروف (25 سلسلة - كل منها 21 حرف بالاضافة إلى حرف صفري أخير) تم نمذجة البرنامج على هيئة نداء لخمس دوال . النداء input (Buffer) يقرأ كل شيء في العازل . النداء tokenize (name, numNames, Buffer) يجزئ العازل ويخزن مؤشرات على اسمائها في المصفوفة name ويعيد عدد الاسماء في numNames . النداء print (name, nunNames) يطبع كل الاسماء المخزنة في العازل . النداء sort (name, numNames) يقوم بترتيب غير مباشر للأسماء المخزنة في العازل بواسطة اعادة ترتيب المؤشرات في المصفوفة name :

```
#include <iostream.h>
#include <string.h>

const int NAME_LENGHT = 20 ;
const int MAX_NUM_NAMES = 25 ;
const int BUFFER_LENGTH = MAX_NUM_NAMES* (NAME_LENGTH + 1) ;
void input (char* buffer) ;
void tokenize (char** name , int& numNames , char* buffer) ;
void print (char** name , int numNames) ;
void sort (char** name, int numNames) ;

main ()
{
    char* name [MAX_NUM_NAMES] ;
    char buffer [BUFFER_LENGTH+1] ;
    int numNames ;
    input (buffer) ;
    tokenize (name, numNames, buffer) ;
    print (name, numNames) ;
    sort (name, numNames) ;
    print (name, numNames) ;
}
```

يتم ادخال كافة الحروف بالنداء cin.getline (buffer, BUFFER_LENGTH, '$') حيث يقرأ الحروف حتى الحرف الدولاري " $ " ويخزن الحروف كلها في العازل .

```
// Reads up to 25 strings into buffer :
void input (char* buffer)
{
    cout << "Enter up to " << MAX_NUM_NAMES << " names, one perline. "
         << " Terminate with \'$\' . \nNames are limited to "
         << NAME_LENGTH << " characters. \n" ;
    cin.getline (buffer, BUFFER_LENGTH, '$') ;
```

الدالة () tokenize تستخدم () strtok لتمر خلال العازل وتجزئ كل سلسلة فرعية تنتهي بعلامة نهاية الخط ' n\' وتخزن عنوانها في المصفوفة name . تواصل الحلقة for حتى تشير p إلى العلامة الدولارية " $ " . لاحظ أن معامل اسم الدالة أعلن عنه كـ **char حيث أنه مصفوفة مؤشرات إلى chars.

لاحظ أيضاً أن العداد n أعلن عنه كـ &int (ارسل بالمرجع) حتى أن قيمته الجديدة تعاد إلى () main.

```
// Copies address of each string in buffer into name array :
void tokenize (char** name, int& n, char* buffer)
{
    char* p = strtok (buffer, " \n ") ;          // p points to each token
    for (n = 0 ; p && *p != '$' ; n++) {
        name [n] = p ;
        p = strtok (NULL, " \n ") ;
    }
}
```

الدوال () print ، () sort مشابهة لما رأيناه من قبل ، فيما عدا أنهما يعملان هنا بصورة غير مباشرة. الدالتان تعملان على المصفوفة name للوصول للاسماء المخزنة في "Buffer" لاحظ أن الدالة () sort تغير فقط المصفوفة name بينما buffer ترك بدون تغيير .

```
// prints the n name stored in buffer :
void print (char** name, int n)
{
    cout << "The names are : \n " ;
    for (int i = 0 ;   i < n ; i++)
        cout << " \t " << i+1 << " . " << name [i] << endl;
}
```

```
//  Sorts the  n  names stored in buffer :
void sort  (char**  name ,  int  n)
{
     char*  temp ;
     for  (int  i  =  1;  i  <  n;  i++)                         //  Bubble sort
          for  (int  j  =  0;  j  <  n-i;  j++)
               if  (strcmp  (name  [j] ,  name  [j+1] )  >  0)  {
                    temp  =  name  [j] ;
                    name  [j]  =  name  [j+1] ;
                    name  [j+1]  =  temp ;
               }
}
```

```
Enter up to 25 names, one per line.  Terminate with '$'.
Names are limited to 20 characters.
Washington, George
Adams, John
Jefferson, Thomas
Madison, James
Monroe, James
Adams, John Quincy
Jackson, Andrew
$The names are:
          1.  Washington, George
          2.  Adams, John
          3.  Jefferson, Thomas
          4.  Madison, James
          5.  Monroe, James
          6.  Adams, John Quincy
          7.  Jackson, Andrew
The names are :
          1.  Adams, John
          2.  Adams, John Quincy
          3.  Jackson, Andrew
          4.  Jefferson, Thomas
          5.  Madison, James
          6.  Monroe, James
          7.  Washington, George
```

في هذه العينة للتنفيذ أدخل المستخدم 7 اسماء ثم علامة الدولار " $ " . بعدها طبعت الأسماء ، رتبت ثم طبعت ثانية.

20.7 اكتب واختبر دالة تعكس سلسلة من الحروف في مكانها دون أي تكرار للحروف.

تحدد الدالة أولاً نهاية السلسلة . ثم تتبادل الحرف الأول مع الأخير ثم الثاني مع قبل الأخير . وهكذا .

```
void reverse (char* s)
{
     for (char* end = s; *end; end ++) ;     // find end of s
     char temp;
     while (s < end - 1) {
          temp = *--end ;
          *end = *s;
          *s++ = temp ;
     }
}
```

يستخدم هذا الاختبار الدالة () getline لقراءة السلسلة ثم تطبعها ثم تعكسها ثم تطبعها مرة ثانية.

```
void reverse (char*) ;
main ()
{
     char string [80] ;
     cin.getline (string, 80) ;
     cout << "The string is [ " << string << " ] . \n" ;
     reverse (string) ;
     cout << "The string is [ " << string << " ] . \n " ;
}
```

```
Today is Wednesday ,
The string is [Today id Wednesday.] .
The string is [.yadsendew si yadot].
```

مسائل برمجة اضافية

21.7 اكتب ونفذ التغيرات للبرنامج في المثال 3.7 الذي يستخدم while (cin >> word) بدلاً من do .. while (*word) .

22.7 اكتب الدالة () strchr .

23.7 اكتب دالة تعيد عدد مرات وجود حرف معطى في سلسلة معينة معطاة .

24.7 اكتب واختبر الدالة () strrlen .

25.7 اكتب واختبر الدالة () strrchr .

26.7 اكتب واختبر الدالة () strstr .

27.7 اكتب واختبر الدالة () strncpy .

28.7 اكتب واختبر الدالة () strcat .

29.7 اكتب واختبر الدالة () strcmp .

30.7 اكتب واختبر الدالة () strncmp .

31.7 اكتب واختبر الدالة () strchr .

32.7 اكتب واختبر الدالة () strrchr .

33.7 اكتب واختبر الدالة () strstr .

34.7 اكتب واختبر الدالة () strspn .

35.7 اكتب واختبر الدالة () strcspn .

36.7 اكتب واختبر الدالة () strpbrk .

37.7 اكتب دالة تعيد عدد الكلمات التي تحتوي على حرف معطى خلال سلسلة معطاة . انظر المثال 19.7.

38.7 اكتب دالة (غير تكرارية) تحدد ما إذا كانت سلسلة معطاة تمثل " باليندروم" (معكوسها يساويها) (انظر المسألة 29.5) .

39.7 حاول التنبؤ ماذا يفعل البرنامج التالي للسلسلة s (انظر المثال 19.7) .نفذ البرنامج لاختبار التنبؤ.

```
#include <iostream.h>
#include <string.h>
// Test-driver for the strtok () function :
main ()
{
    char s [ ] = " ###ABCD#EFG##HIJK#L#MN#####O#P##### " ;
    char* p ;
    cout << "The string is : [ " << s << " ] \nIts tokens are : \n" ;
```

```
    p = strtok (s,  "#") ;
    while  (p)   {
        cout << " \t [ " << p <<  " ] \n " ;
        p = strtok (NULL,  "#") ;
    }
    cout << "The string is :  [ " << s << " ]  \nIts tokens are : \n" ;
}
```

40.7 اكتب برنامج يقرأ سطر واحد من الكتابة ثم يطبعه بجميع حروفه على الصورة الكبيرة capitalized .

41.7 اكتب برنامج يقرأ سطر واحد من الكتابة ثم يطبعها بعد ازالة كل المسافات الخالية بها.

42.7 اكتب برنامج يقرأ سطر واحد من الكتابة ثم يطبع عدد الحروف المتحركة vowels المقروءة.

43.7 اكتب برنامج يقرأ سطر واحد من الكتابة ثم يطبع عدد الكلمات المقروءة .

44.7 اكتب برنامج يقرأ سطر واحد من الكتابة ثم يطبع عدد الكلمات المقروءة وتحتوي على أربع حروف فقط.

45.7 اكتب برنامج يقرأ سطر واحد من الكتابة ثم يطبع نفس الكلمات في ترتيب عكسي على سبيل المثال الدخل :

today is Tuesday

ينتج الخرج

Tuesday is today

46.7 اكتب برنامج ليقرأ سطر واحد من الكتابة ثم يطبعه وكل كلمة معكوسة كمثال على ذلك الدخل:

today is Tuesday

ينتج الخرج

yadot si yadseut

47.7 اكتب البرنامج الذي يقرأ سطر واحد من الكتابة ثم يطبعه بالتغييرات التالية : لكل حدوث لـ "he " تضاف " or she " ; لكل حدوث لـ " him " تضاف " or her" ; لكل حدوث لـ "his" تضاف "or hers" .

48.7 اكتب برنامج يقرأ حتى 50 سطر من الكتابة كل سطر يحتوي على حتى 80 حرف. ثم يطبع كل السطور في ترتيب عكسي . كمثال الدخل :

All in the golden afternoon

Full leisurely we glide ;

For both our oars, with little skill,
By little arms are plied.

ينتج الخرج

By little arms are plied.
For both our oars, with little skill,
Full leisurely we glide ;
All in the golden afternoon

49.7 اكتب برنامج ليقرأ حتى 50 سطر من الكتابة . كل سطر يحتوي حتى 80 حرف ثم يطبع كل الكلمات في كل سطر في ترتيب معاكس . كمثال الدخل :

All in the golden afternoon
Full leisurely we glide ;
For both our oars, with little skill,
By little arms are plied.

ينتج الخرج

afternoon golden the in All
skill, little with oars, our both For
glide; we leisurely Full
plide. are arms little By

50.7 اكتب برنامج يقرأ حتى 50 سطر من الكتابة . كل سطر يحتوي حتى 80 حرف . وعندها يطبع كل الكلمات في كل سطر بترتيب ابجدي . كمثال ، الدخل :

All in the golden afternoon
Full leisurely we glide ;
For both our oars, with little skill,
By little arms are plied.

ينتج الخرج

afternoon All golden in the
Full glide; leisurely we

both For little oars, our skill, with

are arms By little plied.

51.7 اكتب برنامج يقرأ حتى 50 سطر من الكتابة كل سطر يحتوي حتى 80 حرف وعندها يعيد تشكيل الكتابة بحيث لا يحتوي أي سطر على أكثر من 40 حرف كمثال

"The first thing I 've got to do, " said Alice to herself, as she wandered about in the wood, "is to grow to my right size again; and the second thing is to find my way into that lovely garden.

ينتج الخرج

"The first thing I 've got to do, " said
Alice to herself, as she wandered about
in the wood, "is to grow to my right size
again; and the second thing is to find my
way into that lovely garden.

52.7 اكتب برنامج يشفر وبعدها يفك شفرة سطر من الكتابة يجب أن يدخل البرنامج shift key k للتشفير : هذا يكون عدد صحيح من 1 إلى 25 . عندها يقرأ البرنامج سطر من الكتابة ، يطبعه ، يشفره ، يطبع حروف الشفرة ، يفك شفرتها بعدها يطبع الحروف العادية ليبين أنها نفس الكتابة الاصلية. التشفير وفك التشفير يجب أن يتم بدالتين منفصلتين . ببساطة يشفر الحرف بإضافة حرف k اليه ويفك الشفرة بطرح k منه. العمليتين يجب أن يدورا حول نهاية الحروف الابجدية كمثال على ذلك الحرف 'w' يشفر إلى 'B' و 'D' تفك شفرتها إلى 'y' إذا كانت k = 6 .

53.7 اكتب برنامج يلعب لعبة "هانجمان"

54.7 اكتب دالة تطبع جملة عشوائية . استعمل المصفوفات التالية :

```
char*  article [5]  =  { "a" , "some" , "that" , "this" , "the" }
char*  noun [5]  =  { "boy" , "dog" , "girl" , "man" , "woman" };
char*  verb [5]  =  { "barked at" ,  "bit" , "kissed" , "spoke to" } ;
```

55.7 اكتب واختبر الدالة التالية التي تتابع التكرارية لكل من الـ 26 حرف (دون النظر لحجم الحرف) في السلسلة التالية :

```
void tally  (int  frequency [ ] ,   const char*  s)
```

56.7 اكتب واختبر الدالة التالية والتي تحذف الحروف المكررة في السلسلة المعطاة :

```
void delDups (char* s)
```

كمثال ، إذا كانت s هي السلسلة "ABRACADABRA" فإنه بعد نداء delDups (s) تخفض السلسلة إلى "ABRCD" .

57.7 اكتب واختبر الدالة التالية والتي تحذف من s1 كل وجود للحروف التي توجد في s2 :

```
void del (char* s1, const char* s2)
```

كمثال: إذا كانت s1 هي السلسلة "ABRACADABRA" والسلسلة s2 هي "AB" فبعد النداء del (s1 , s2) تخفض السلسلة s1 إلى "RCDR".

إجابات أسئلة المراجعة

1.7 من بين الثلاثة عشر اعلاناً

أ – ما يلي هي اعلانات جائزة في لغة ++C لسلاسل الحروف :

```
char s [6] ;
char s [6] = { 'H' , 'e' ,  'l',  'l',  'o' } ;
char s [6] = "Hello" ;
char s [] = { 'H' , 'e' ,  'l',  'l',  'o' } ;
char s [] = "Hello" ;
char* s;
char* s = new char [6] ;
char* s = "Hello" ;
```

ب – ما يلي هي اعلانات صحيحة لسلسلة الحروف في ++C التي طولها 5 . بدأت السلسلة بـ "Hello" وتخصصت في وقت الترجمة.

```
char s [6] = { 'H' , 'e' ,  'l',  'l',  'o' } ;
char s [6] = "Hello" ;
char s [] = { 'H' , 'e' ,  'l',  'l',  'o' } ;
char s [] = "Hello" ;
char* s = "Hello" ;
```

جـ – لا يمكن تحديد سلسلة مثل هذه في وقت التنفيذ .

د – ما يلي اعلانات سارية لسلسلة الحروف في C++ كثابت رسمي لدالة :

```
char s [] ;
char* s ;
```

2.7 هذا يقرأ فقط حتى أول مسافة بيضاء . للدخل المعطى فإنه ينسب "Hello," إلى s .

3.7 هذه الشفرة تعد عدد الحروف الكبيرة في السلسلة s . ولذلك فالخرج هو 6 .

4.7 هذه الشفرة تحول كل الحروف الكبيرة إلى حروف صغير في السلسلة s :

```
123 w.  42nd  st .,  ny ,  ny 10020-1095
```

لاحظ أنه لتحويل حالة الحرف p* لابد من أن تنسب اليه قيمة العودة للدالة (*p = tolower (*p .

5.7 هذا الأمر يزيد كل الحروف الكبيرة . يغيرW إلىX و S إلى T وهكذا.

```
123X.  42nd  Tt. ,  OZ,  OZ   10020-1095
```

6.7 هذه الشفرة تعد الحروف المنقوطة في السلسلة s ، ولهذا فالخرج هو 5 .

7.7 هذا يستبدل كل حرف بعده حرف تنقيط بالحرف التالي .

```
123 ..  42nd  s .,  ,  N , ,  NY   1002-1095
```

8.7 التنسيب s1 = s2 تجعل ببساطة s1 مسمى آخر لـ s2 ، أي يشير كلاهما إلى نفس السلسلة. النداء strcpy (s1, s2) تنسخ الحروف في s2 إلى s1 لذلك فإنها تضاعف السلسلة.

9.7

أ – هذا ينسب العدد الصحيح 0 إلى n

ب – هذا ينسب السلسلة الجزئية "rford" إلى s1 .

جـ – هذا ينسب السلسلة الجزئية "rd" إلى s1 .

د – هذا ينسب السلسلة الجزئية "utherford" إلى s1 .

هـ – هذا ينسخ last إلى first حيث first سيكون ايضاً السلسلة "Hayes" .

و – هذا ينسخ السلسلة الفرعية "Hay" في الجزء الأول في first لتصبح "Hayherford" .

ز – هذا يضم last لنهاية first لتصبح "RutherfordHayes" .

ح – هذا يضم السلسلة الفرعية "Hay" لنهاية first لتصبح "RutherfordHay".

10.7

أ - 7

ب - 6

جـ - 5

د - 7

11.7 تطبع ABCDE >= ABC

12.7 تطبع ABCDE < ABCE

13.7 تطبع ABCDE >=

14.7 تطبع =!

الملاحـــــق

الملحق A

شفرات ASCII

كل حرف يخزن في صورة شفرات ASCII، وهذه الشفرات عبارة عن أرقام صحيحة من صفر حتى 127. أول 32 حرف هى عبارة عن أحرف غير مكتوبة، لذلك فإن رموزهم فى العمود الأول تكون مصحوبة إما بالأحرف"Ctrl" (وتنطق كونترول) أو بالحرف '\' " وتنطق (باك سلاش) أو الشرطة العكسية. فى الحالة الأولى نضغط المفتاح كونترول ctrl مع المفتاح الثانى للحصول على الحرف الغير مكتوب. فمثلاً حرف نهاية الملف (end-of-file) والذى شفرته هى 4 يتم إدخاله بالضغط على الزرار ctrl وفى نفس الوقت نضغط الزرار D (ومع بعضها تنطق كونترول D) .

فى الحالة الثانية نكتب الحرف '\' متبوعاً بالحرف المطلوب، فمثلاً فى لغة ++C الشفرة " n\ " هى شفرة السطر الجديد newline وشفرتها هى 10.

Character	Description	Decimal	Octal	Hex	Binary
Ctrl - @	Null, end of string	0	0	0x0	0000 0000
Ctrl - A	Start of heading	1	01	0x1	0000 0001
Ctrl - B	Start of text	2	02	0x2	0000 0010
Ctrl - C	End of text	3	03	0x3	0000 0011
Ctrl - D	End of transmission, end of file	4	04	0x4	0000 0100
Ctrl - E	Enquiry	5	05	0x5	0000 0101
Ctrl - F	Acknowledge	6	06	0x6	0000 0110
\a	Bell, alert, System beep	7	07	0x7	0000 0111
\b	Backspace	8	010	0x8	0000 1000
\t	Horizontal tab	9	011	0x9	0000 1001
\n	Line feed, new line	10	012	0xa	0000 1010
\v	Vertical tab	11	013	0xb	0000 1011
\f	Form feed, new page	12	014	0xc	0000 1100
\r	Carriage return	13	015	0xd	0000 1101
Ctrl - N	Shift out	14	016	0xe	0000 1110
Ctrl - O	Shift in	15	017	0xf	0000 1111
Ctrl - P	Data link escape	16	020	0x10	0001 0000
Ctrl - Q	Device control 1, resume scroll	17	021	0x11	0001 0001

1- ASCII هى اختصار للعبارة (الشفرات الأمريكية القياسية لتبادل المعلومات) أو
American Standard Code for Information Interchange

تابع شفرات ASCII

Character	Description	Decimal	Octal	Hex	Binary
Ctrl - R	Devic control 2	18	022	0x12	0001 0010
Ctrl - S	Device control 3, stop scroll	19	023	0x13	0001 0011
Ctrl - T	Device control 4	20	024	0x14	0001 0100
Ctrl - U	Negative acknowledgment	21	025	0x15	0001 0101
Ctrl - V	Synchronous idle	22	026	0x16	0001 0110
Ctrl - W	End transmission block	23	027	0x17	0001 0111
Ctrl - X	Cancel	24	030	0x18	0001 1000
Ctrl - Y	End of message, interrupt	25	031	0x19	0001 1001
Ctrl - Z	Substitute, exit	26	032	0x1a	0001 1010
Ctrl - [	Escape	27	033	0x1b	0001 1011
Ctrl - /	File separator	28	034	0x1c	0001 1100
Ctrl -]	Group separator	29	035	0x1d	0001 1101
Ctrl - ^	Record separator	30	036	0x1e	0001 1110
Ctrl - _	Unit separator	31	037	0x1f	0001 1111
	Blank, space	32	040	0x20	0010 0000
!	Exclamation point	33	041	0x21	0010 0001
"	Quotation mark, double quote	34	042	0x22	0010 0010
#	Hash mark, number sign	35	043	0x23	0010 0011
$	Dollar sign	36	044	0x24	0010 0100
%	Percent sign	37	045	0x25	0010 0101
&	Ampersand	38	046	0x26	0010 0110
'	Apostrophe, single quote	39	047	0x27	0010 0111
(	Left parenthesis	40	050	0x28	0010 1000
)	Right parenthesis	41	051	0x29	0010 1001
*	Asterisk, star, times	42	052	0x2a	0010 1010
+	Plus	43	053	0x2b	0010 1011
,	Comma	44	054	0x2c	0010 1100
-	Dash, minus	45	055	0x2d	0010 1101
.	Dot. period, decimal point	46	056	0x2e	0010 1110
/	Slash	47	057	0x2f	0010 1111
0	Digit Zero	48	060	0x30	0011 0000

تابع شفرات ASCII

Character	Description	Decimal	Octal	Hex	Binary
1	Digit one	49	061	0x31	0011 0001
2	Digit two	50	062	0x32	0011 0010
3	Digit three	51	063	0x33	0011 0011
4	Digit four	52	064	0x34	0011 0101
5	Digit five	53	065	0x35	0011 0100
6	Digit six	54	066	0x36	0011 0110
7	Digit seven	55	067	0x37	0011 0111
8	Digit eight	56	070	0x38	0011 1000
9	Digit nine	57	071	0x39	0011 1001
:	Colon	58	072	0x3a	0011 1010
;	Semicolon	59	073	0x3b	0011 1011
<	Less than	60	074	0x3c	0011 1100
=	Equal to	61	075	0x1d	0011 1100
>	Greater than	62	076	0x3e	0011 1110
?	Question mark	63	077	0x3f	0011 1111
@	Commercial at sign	64	0100	0x40	0100 0000
A	Letter capital A	65	0101	0x41	0100 0001
B	Letter capital B	66	0102	0x42	0100 0010
C	Letter capital C	67	0103	0x43	0100 0011
D	Letter capital D	68	0104	0x44	0100 0100
E	Letter capital E	69	0105	0x45	0100 0101
F	Letter capital F	70	0106	0x46	0100 0110
G	Letter capital G	71	0107	0x47	0100 0111
H	Letter capital H	72	0110	0x48	0100 1000
I	Letter capital I	73	0111	0x49	0100 1001
J	Letter capital J	74	0112	0x4a	0100 1010
K	Letter capital K	75	0113	0x4b	0100 1011
L	Letter capital L	76	0114	0x4c	0100 1100
M	Letter capital M	77	0115	0x4d	0100 1101
N	Letter capital N	78	0116	0x4e	0100 1110
O	Letter capital O	79	0117	0x4f	0100 1111

تابع شفرات ASCII

Character	Description	Decimal	Octal	Hex	Binary
P	Letter capital P	80	0120	0x50	0101 0000
Q	Letter capital Q	81	0121	0x51	0101 0001
R	Letter capital R	82	0122	0x52	0101 0010
S	Letter capital S	83	0123	0x53	0101 0011
T	Letter capital T	84	0124	0x54	0101 0100
U	Letter capital U	85	0125	0x55	0101 0101
V	Letter capital V	86	0126	0x56	0101 0110
W	Letter capital W	87	0127	0x57	0101 0111
X	Letter capital X	88	0130	0x58	0101 1000
Y	Letter capital Y	89	0131	0x59	0101 1001
Z	Letter capital Z	90	0132	0x5a	0101 1010
[	Left braket	91	0133	0x5b	0101 1011
\	Backslash	92	0134	0x5c	0101 1100
]	Right braket	93	0135	0x5d	0101 1101
^	Caret	94	0136	0x5e	0101 1110
-	Underscore	95	0137	0x5f	0101 1111
'	Accent grave	96	0140	0x60	0110 0000
a	Letter lowercase A	97	0141	0x61	0110 0001
b	Letter lowercase B	98	0142	0x62	0110 0010
c	Letter lowercase C	99	0143	0x63	0110 0011
d	Letter lowercase D	100	0144	0x64	0110 0100
e	Letter lowercase E	101	0145	0x65	0110 0101
f	Letter lowercase F	102	0146	0x66	0110 0110
g	Letter lowercase G	103	0147	0x67	0110 0111
h	Letter lowercase H	104	0150	0x68	0110 1000
i	Letter lowercase I	105	0151	0x69	0110 1001
j	Letter lowercase J	106	0152	0x6a	0110 1010
k	Letter lowercase K	107	0153	0x6b	0110 1011
l	Letter lowercase L	108	0154	0x6c	0110 1100
m	Letter lowercase M	109	0155	0x6d	0110 1101
n	Letter lowercase N	110	0156	0x6e	0110 1110

تابع شفرات ASCII

Character	Description	Decimal	Octal	Hex	Binary
o	Letter lowercase O	111	0157	0x6f	0110 1111
p	Letter lowercase P	112	0160	0x70	0111 0000
q	Letter lowercase Q	113	0161	0x71	0111 0001
r	Letter lowercase R	114	0162	0x72	0111 0010
s	Letter lowercase S	115	0163	0x73	0111 0011
t	Letter lowercase T	116	0164	0x74	0111 0110
u	Letter lowercase U	117	0165	0x75	0111 0101
v	Letter lowercase V	118	0166	0x76	0111 0110
w	Letter lowercase W	119	0167	0x77	0111 0111
x	Letter lowercase X	120	0170	0x78	0111 1000
y	Letter lowercase Y	121	0171	0x79	0111 1001
z	Letter lowercase Z	122	0172	0x7a	0111 1010
{	Left brace	123	0173	0x7b	0111 1011
|	Pipe	124	0174	0x7c	0111 1100
}	Right brace	125	0175	0x7d	0111 1101
~	Tilde	126	0176	0x7e	0111 1110
Delete	Delete, rub out	127	0177	0x7f	0111 1111

الملحق B

الكلمات المفتاحية في لغة C++

لغة C++ لها 48 كلمة مفتاحية، هذه الكلمات الخاصة تستخدم لتحديد التركيب اللغوى لهذه اللغة.

Keyword	Description	Example
asm	Allows information to be passed to the assembler directly يسمح بتمرير المعلومات إلى الاسمبلر مباشرة	asm ("check");
auto	Storage class for objects that exist only within their own bloch طبقة تخزين للأهداف الموجودة خلال البلوك الخاص بهم	auto int n;
break	Terminates a loop or a switch statement نهاية حلقة أو الأمر swich	break;
case	Used in a switch statement to specify control expression تستخدم مع الأمر switch لتحديد أمر التحكم	switch (n/10)
catch	Specifies actions to take when an exception occurs تحدد فعل معين يجب أخذه عند حدوث استثناء معين	catch (error)
char	An integer type نوع البيانات الصحيح (أو الحرفي)	char c;
class	Specifies a class declaration يحدد اعلان طبقة	class X (...);
const	Specifies a constant definition يحدد اعلان ثابت	const int s = 32;
continue	Jumps to beginning of next iteration in a loop القفز لبداية المحادثة التالية في حلقة	continue;
default	The "otherwise" case in a switch statement الحالة التلقائية في الأمر switch .	default : sum = 0;
delete	Deallocates memory allocated by a new statement اعادة تحرير ذاكرة تم تحديدها بالأمر New	delete a;
do	Specifies a do..while loop يحدد الحلقة do ... while	do { ... } while ...
double	A real number type نوع الأرقام الحقيقية	double x;
else	Specifies alternative in an if statement تحدد الخيار للأمر if	else n = 0;

تابع الكلمات المفتاحية ++C

Keyword	Description	Example
enum	Used to declare an enumeration type تستخدم للاعلان عن الأنواع المتعددة	enum tool { ... };
extern	Storage class for objects declared outside the local block طبقة تخزين للأهداف المعلنة خارج نطاق البلوك المحلي	extern int max;
float	A real number type نوع للأرقام الحقيقية	dloat x;
for	Specifies a for loop تحدد الحلقة for	for (; ;) ...
friend	Specifies a friend function in a class تحدد دالة صديقة في طبقة	friend int f ( );
goto	Causes execution to jump to a labeled statement يسبب القفز بعملية التنفيذ إلى مكان له علامة	goto error;
if	Specifies an if statement تحدد الأمر if	if (n > o) ...
inline	Declares a function whose text is to be substituted for its call تحدد دالة يتم احلال نصها عند النداء عليها	inline int f ( );
int	An integer type نوع الأرقام الصحيحة	int n;
long	Used to define integer and real types يحدد أرقام صحيحة وحقيقية	long double x;
new	Allocates memory يحجز ذاكرة	int* p = new int;
operator	Used to declare an operator overload يستخدم للاعلان عن زيادة تحميل معامل	X operator ++ ( );
private	Specifies private declarations in a class يحدد اعلانات خاصة في طبقة	private: int n;
protected	Specifies protected declarations in a class يحدد اعلانات محمية في طبقة	protected: int n;
public	Specifies public declarations in a class يحدد اعلانات عامة في طبقة	public: int n;
register	Storage class specifier for objects stored in registers محدد تخزين لتحديد أهداف تخزن في مسجل	register int i;

تابع الكلمات المفتاحية ++C

Keyword	Description	Example
return	Statement that terminates a function and returns a value أمر ينهي دالة ويعود بقيمة	return 0;
short	An integer type نوع أرقام صحيحة	short n;
signed	Used to define integer types يستخدم لتحديد ارقام صحيحة	signed char c;
sizeof	Operator that returns the number of bytes used to store an objects معامل يعود بعدد البانيات المستخدمة في تخزين الهدف .	n = sizeof (float;
static	Storage class of objects that exist for the duration of the program نوع تخزين يكون موجود أو معرف طالما البرنامج موجود .	static int n;
struct	Specifies a structure definition اعلان تحديد هيكل	struct X { ... };
switch	Specifies a switch statement يحدد الامر swich	switch (n) { ... };
template	Specifies a template class يحدد طبقة نموذج	template <class T>
this	Pointer that points to the current object مؤشر للهدف الحالي	return *this;
throw	Used to generate an exception يستخدم لتوليد استثناء	throw X ( );
try	Specifies a block that contains exception handlers تحدد بلوك يحتوي على مناول استثناء	try { ... }
typedef	Declares a synonym for an existing type يعلن عن مرادف لنوع موجود	typedef int Num;
union	Specifies a structure whose elements occupy the same storage يحدد هيكل تحتل كل عناصره نفس المخزن .	\union z { ... } ;
unsigned	Used to define integer types يستخدم لتعريف أنواع أرقام صحيحة	unsigned int b;
virtual	Declares a member function that is defined in a subclass يعلن عن دالة عضو محددة في طبقة فرعية	virtual int f ( );
void	Designates the absence of a type يميز غياب النوع	void f ( );

تابع الكلمات المفتاحية ++C

Keyword	Description	Example
volatile	Declares objects that can be modified outside of program control يعلن عن أهداف يمكن تعديلها خارج نطاق تحكم البرنامج.	int volatile n;
while	Specifies a while loop يحدد الحلقة while	while (n > 0) ...

الملحق C

العمليات فى لغة ++C

هذا الجدول يبين جميع العمليات فى لغة ++C مرتبة على حسب أسبقية العمليات حيث العمليات ذات الأسبقية الأعلى تنفيذ قبل العمليات ذات الأسبقية الأقل. فمثلاً فى العلاقة (a - b*c)، عملية الضرب * ستنفذ أولاً ثم عملية الطرح ثانيا لأن عملية الضرب لها أسبقية أعلى (13) من عملية الطرح (12) . العمود المعنون "Assoc." (وهى اختصار لكلمة إلحاق Association) يخبرنا إذا كانت العملية تلحق من اليمين أم من اليسار، فمثلاً العلاقة (a- b*c) ستنفذ كالتالى (a- b) - c)) لأن عملية الطرح تلحق من اليسار . أما العمود المعنون "Arity" فيخبرنا عما اذا كانت هذه العملية تجرى على معامل واحد أم معاملان أم ثلاثة. العمود المعنون "Ovrldbl." يخبرنا إذا كانت هذه العملية يمكن زيادة تحميلها أم لا (أنظر فصل 8) .

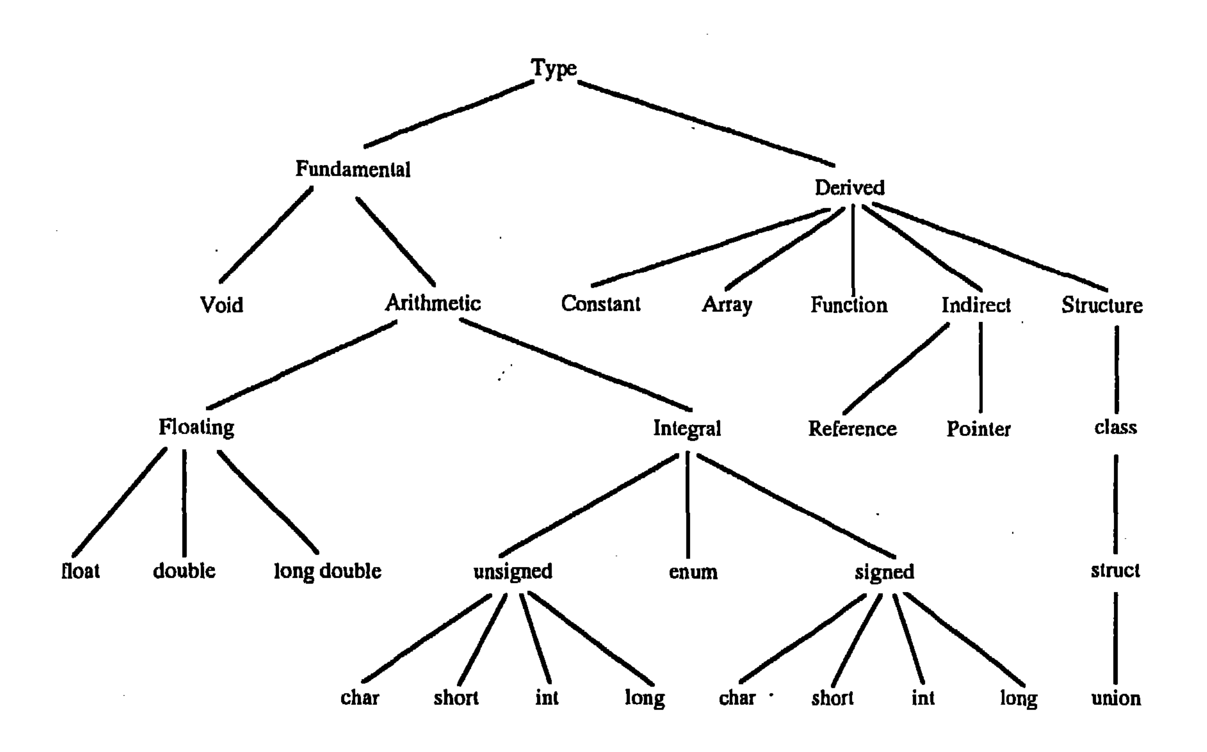
Op	Name	Prec.	Assoc.	Arity	Ovrldbl.	Example
: :	Global scope resolution	17	Right	Unary	No	: : x
: :	Class scope resolution	17	Left	Binary	No	x : : x
.	Direct member selection	16	Left	Binary	No	S. len
->	Indirect member selection	16	Left	Binary	Yes	p-> len
[ ]	Subscript	16	Left	Binary	Yes	a [i]
( )	Function call	16	Left	n\a	Yes	rand ( )
( )	Type construction	16	Left	n/a	Yes	int (ch)
++	Post-increment	16	Right	Unary	Yes	n++
- -	Post-decrement	16	Right	Unary	Yes	n--
sizeof	Size of object or type	15	Right	Unary	No	sixeof (a)
++	Pre-increment	15	Right	Unary	Yes	++n
- -	Pre-decrement	15	Right	Unary	Yes	--n
~	Bitwise complement	15	Right	Unary	Yes	~s
!	Logical NOT	15	Right	Unary	Yes	! p
+	Unary plus	15	Right	Unary	Yes	+n
-	Unary minus	15	Right	Unary	Yes	-n
*	Dereference	15	Right	Unary	Yes	*p
&	Address	15	Right	Unary	Yes	& x
new	Allocation	15	Right	Unary	Yes	new p
delete	Deallocation	15	Right	Unary	Yes	delete p
( )	Type conversion	15	Right	Binary	Yes	int (ch)
. *	Direct membeer selection	14	Left	Binary	No	x. * q

أسبقية العمليات في لغة ++C

Op	Name	Prec.	Assoc.	Arity	Ovrldbl.	Example
->*	Indirect membeer selection	14	Left	Binary	Yes	p-> q
*	Multiplication	13	Left	Binary	Yes	m*n
/	Division	13	Left	Binary	Yes	m/n
%	Remainder	13	Left	Binary	Yes	m%n
+	Addition	12	Left	Binary	Yes	m + n
-	Subtraction	12	Left	Binary	Yes	m - n
<<	Bit shift left	11	Left	Binary	Yes	cout << n
>>	Bit shift right	11	Left	Binary	Yes	cin >> n
<	Less than	10	Left	Binary	Yes	x < y
<=	Less than or equal to	10	Left	Binary	Yes	x <= y
>	Greater than	10	Left	Binary	Yes	x > y
>=	Greater than or equal to	10	Left	Binary	Yes	x >= y
= =	Equal to	9	Left	Binary	Yes	x = = y
! =	Not equal to	9	Left	Binary	Yes	x ! = y
&	Bitwise AND	8	Left	Binary	Yes	s&t
^	Bitwise XOR	7	Left	Binary	Yes	s^t
|	Bitwise OR	6	Left	Binary	Yes	s | t
&&	Logical AND	5	Left	Binary	Yes	u && v
||	Logical OR	4	Left	Binary	Yes	u || v
? :	Conditional expression	3	Left	Tamary	No	u ? x : y
=	Assignment	2	Right	Binary	Yes	n = 22
+ =	Addition assignment	2	Right	Binary	Yes	n + = 8
- =	Subtraction assignment	2	Right	Binary	Yes	n - = 4
* =	Multiplication assignment	2	Right	Binary	Yes	n * = -1
/ =	Division assignment	2	Right	Binary	Yes	n \ = 10
% =	Remainder assignment	2	Right	Binary	Yes	n %= 10
& =	Bitwise AND assignment	2	Right	Binary	Yes	s &= mask
^ =	Bitwise XOR assignment	2	Right	Binary	Yes	s ^= mask
| =	Bitwise OR assignment	2	Right	Binary	Yes	s |= mask
<<=	Bit shift left assignment	2	Right	Binary	Yes	s <<= 1
>>=	Bit shift right assignment	2	Right	Binary	Yes	s >>= 1
throw	Throw exception	1	Right	Unary	Yes	throw
,	Comma	0	Left	Binary	Yes	++m, --n

الملحق D

الأنواع في لغة ++C



الملحق E

المراجع

[Adams]

C++ An Introduction to Computing, by Joel Adams, Sanford Leestma, and Larry Nyhoff. Prentice Hall, Englewood Cliffs, NJ (1995) 0-02-369402-5.

[Barton]

Scientific and Engineering C++, by John J. Barton and Lee R. Nackman. Addison-Wesley Publishing Company, Reading, MA (1994) 0-201-53393-6.

[Bergin]

Data Abstraction, the Object-Oriented Approach Using C++, by Joseph Bergin. McGraw-Hill, Inc., New York, NY (1994) 0-07-911691-4.

[Bronson]

A First Book of C++, by Gary J. Bronson. West Publishing Company, St. Paul, MN (1995) 0-314-04236-9.

[Budd]

Classic Data Structures in C++, by Timothy A. Budd. Addison-Wesley Publishing Company, Reading, MA (1994) 0-201-50889-3.

[Capper]

Introducing C++ for Scientists, Engineers and Mathematicians, by D. M. Capper. Springer-Verlag, London (1994) 3-540-19847-4.

[Cargill]

C++ Programming Style, by Tom Cargill. Addison-Wesley Publishing Company, Reading, MA (1992) 0-201-56365-7.

[Carrano]

Data Abstraction and Problem Solving with C++, by Frank M. Carrano. Benjamin/Cummings Publishing Company, Redwood City, CA (1993) 0-8053-1226-9.

[Carroll]

Designing and Coding Reusable C++, by Martin D. Carroll and Margaret A. Ellis. Addison-Wesley Publishing Company, Reading, MA (1995) 0-201-51284-X.

[Cline]

C++ FAQs, by Marshall P. Cline and Greg A. Lomow. Addison-Wesley Publishing Company, Reading, MA (1995) 0-201-58958-3.

[Coplien]

Advanced C++, Programming Styles and Idioms, by James O. Coplien. Addison-Wesley Publishing Company, Reading, MA (1992) 0-201-54855-0.

[Deitel]

C++ How to Program, by H. M. Deitel and P. J. Deitel.
Prentice Hall, Englewood Cliffs, NJ (1994) 0-13-117334-0.

[Dewhurst]

Programming in C++, Second Edition, by Stephen C. Dewhurst and Kathy T. Stark.
Prentice Hall, Englewood Cliffs, NJ (1995) 0-13-182718-9.

[Ellis]

The Annotated C++ Reference Manual, by Margaret A. Ellis and Bjarne Stroustrup.
Addison-Wesley Publishing Company, Reading, MA (1992) 0-201-51459-1.

[Friedman]

Problem Solving, Abstraction, and Design Using C++, by F. L. Friedman and E. B. Koffman.
Addison-Wesley Publishing Company, Reading, MA (1994) 0-201-52649-2.

[Graham]

Learning C++, by Neill Graham.
McGraw-Hill, Inc, New York, NY (1991) 0-07-023983-5.

[Hansen]

The C++ Answer Book, by Tony L. Hansen.
Addison-Wesley Publishing Company, Reading, MA (1990) 0-201-11497-6.

[Headington]

Data Abstraction and Structures Using C++, by Mark R. Headington and David D. Riley.
D. C. Heath and Company, Lexington, MA (1994) 0-669-29220-6.

[Horowitz]

Fundamentals of Data Structures in C++, by Ellis Horowitz, Sartaj Sahni, and Dinesh Mehta.
W. H. Freeman and Company, New York, NY (1995) 0-7167-8292-8.

[Johnsonbaugh]

Object-Oriented Programming in C++, by Richard Johnsonbaugh and Martin Kalin.
Prentice Hall, Englewood Cliffs, NJ (1995) 0-02-360682-7.

[Knuth1]

The Art of Computer Programming, Volume 1: Fundamental Algorithms, Second Edition,
by Donald E. Knuth.
Addison-Wesley Publishing Company, Reading, MA (1973) 0-201-03809-9.

[Knuth2]

The Art of Computer Programming, Volume 2: Seminumerical Algorithms, Second Edition,
by Donald E. Knuth.
Addison-Wesley Publishing Company, Reading, MA (1981) 0-201-03822-6.

[Knuth3]

The Art of Computer Programming, Volume 3: Sorting and Searching, by Donald E. Knuth.
Addison-Wesley Publishing Company, Reading, MA (1973) 0-201-03803-X.

[Ladd]

C++ Templates and Tools, by Scott Robert Ladd.
M&T Books, New York, NY (1995) 0-55851-437-6.

[Lippman]

The C++ Primer, Second Edition, by Stanley B. Lippman.
Addison-Wesley Publishing Company, Reading, MA (1991) 0-201-54848-8.

[Meyers]

Effective C++, by Scott Meyers.
Addison-Wesley Publishing Company, Reading, MA (1992).

[Model]

Data Structures, Data Abstraction: A Contemporary Introduction Using C++, by M. L. Model.
Prentice Hall, Englewood Cliffs, NJ (1994) 0-13-088782-X.

[Murray]

C++ Strategies and Tactics, by Robert B. Murray.
Addison-Wesley Publishing Company, Reading, MA (1993) 0-201-56382-7.

[Nagler]

Learning C++, by Eric Nagler.
West Publishing Company, St. Paul, MN (1993) 0-314-02464-6.

[Nelson]

C++ Programmers Guide to the Standard Template Library, by Mark Nelson.
IDG Books Worldwide, Inc., Foster City, CA (1995) 0-56884-314-3.

[Oualline]

Practical C++ Programming, by Steve Oualline.
O'Reilly & Associates, Sebastopol, CA (1995) 1-56592-139-9.

[Perry]

An Introduction to Object-Oriented Design in C++, by Jo Ellen Perry and Harold D. Levin.
Addison-Wesley Publishing Company, Reading, MA (1996) 0-201-76564-0.

[Plauger1]

The Standard C Library, by P. J. Plauger.
Prentice Hall, Englewood Cliffs, NJ (1992) 0-13-131509-9.

[Plauger2]

The Draft Standard C++ Library, by P. J. Plauger.
Prentice Hall, Englewood Cliffs, NJ (1995) 0-13-117003-1.

[Pohl.1]

Object-Oriented Programming Using C++, by Ira Pohl.
The Benjamin/Cummings Publishing Company, Inc, Redwood City, CA (1993) 0-8053-5384-4.

[Pohl.2]

C++ for Pascal Programmers, Second Edition, by Ira Pohl.
The Benjamin/Cummings Publishing Company, Inc, Redwood City, CA (1994) 0-8053-3158-1.

[Prata]

C++ Primer Plus, by Stephen Prata.
Waite Group Press, Corte Madera, CS (1991) 0-878739-02-6.

[Ranade & Zamir]

C++ Primer for C Programmers, by Jay Ranade and Saba Zamir.
McGraw-Hill, Inc., New York, NY (1994) 0-07-051487-9.

[Rudd]
C++ Complete, by Anthony Rudd.
John Wiley & Sons, Inc, New York, NY (1994) 0-471-06565-X.

[Satir]
C++: The Core Language, by Gregory Satir and Doug Brown.
O'Reilly & Associates, Sebastopol, CA (1995) 0-56592-116-X.

[Savitch]
Problem Solving with C++, by Walter Savitch.
Addison-Wesley Publishing Company, Reading, MA (1996) 0-8053-7440-X.

[Sedgewick]
Algorithms in C++, by Robert Sedgewick.
Addison-Wesley Publishing Company, Reading, MA (1992) 0-201-51059-6.

[Sengupta]
C++ Object-Oriented Data Structures, by Saumyendra Sengupta and Carl Phillip Korobkin.
Springer-Verlag, New York, NY (1994) 0-387-94194-0

[Sessions]
Class Construction in C and C++, by Roger Sessions.
PTR Prentice Hall, Englewood Cliffs, NJ (1992) 0-13-630104-5.

[Shammas]
Advanced C++, by Namir Clement Shammas.
SAMS Publishing, Carmel, IN (1992) 0-672-30158-X.

[Stepanov]
"The Standard Template Library," *Technical Report HPL-94-34*, by A. A. Stepanov and M. Lee.
Hewlett-Packard Laboratories, April 1994.

[Stroustrup1]
The C++ Programming Language, Second Edition, by Bjarne Stroustrup.
Addison-Wesley Publishing Company, Reading, MA (1991) 0-201-53992-6.

[Stroustrup2]
The Design and Evolution of C++, by Bjarne Stroustrup.
Addison-Wesley Publishing Company, Reading, MA (1994) 0-201-54330-3.

[Teale]
C++ IOStreams, by Steve Teale.
Addison-Wesley Publishing Company, Reading, MA (1993) 0-201-59641-5.

[Wang]
C++ with Object-Oriented Programming, by Paul S. Wang.
PWS Publishing Company, Boston, MA (1994) 0-534-19644-6.

[Weiss]
Data Structures and Algorithm Analysis in C++, by Mark Allen Weiss.
Benjamin/Cummings Publishing Company, Redwood City, CA (1994) 0-8053-5443-3.

[Winston]
On to C++, by Patrick Henry Winston.
Addison-Wesley Publishing Company, Reading, MA (1994) 0-201-58043-8.

الملحق F

الدوال سابقة التعريف

هذا الملحق يصف الدوال سابقة التعريف الموجودة فى مكتبة ++C . العمود الأول يحتوى اسم الدالة، والثانى يحتوى النوع الأولى لهذه الدالة ووصف مختصر لما تفعله والعمود الثالث يبين ملف الرأس المعرفة فيه هذه الدالة.

Function	Prototype and Description	Header File
abort ( )	void abort ( ) ; خروج من البرنامج	<stdlib. h>
abs ( )	int abs (int n) ; يعود بالقيمة المطلقة لـ n	<stdlib. h>
acos ( )	double acos (double x) ; يعود بعكس الـ cosine	<math. h>
asin ( )	double asin (double x) ; يعود بعكس الـ sin	<math. h>
atan ( )	double atan (double x) ; يعود بعكس tangent	<math. h>
atof ( )	double atof (const char* s); يعود بالرقم الحقيقي الممثل حرفياً في السلسلة S	<stdlib. h>
atoi ( )	int atoi (const char* s) ; يعود بالرقم الصحيح الممثل في السلسلة S	<stdlib. h>
atol ( )	long atol (const char* s) ; يعود بالرقم الصحيح الممثل في السلسلة S	<stdlib. h>
bad ( )	int ios : : bad ( ) ; يعود بقيمة غير الصفر إذا كانت badbit تساوي صفر ، ويعود بصفر اذا كانت غير ذلك	<iostream. h>
bsearch ( )	void* bsearch (const void* x, void* a, size_t n, size_t s, int (*cmp) (const void*, const void*)) ;	<stdlib. h>

Function	Prototype and Description	Header File
	ينفذ خوارزم البحث الثنائي للبحث عن x في الصف المرتب a الذي يحتوي n من العناصر كل منها له حجم S مستخدماً comp* لمقارنة أي عنصرين. إذا وجد العنصر يعود بمؤشر إليه ، وإذا لم يوجد يعود المؤشر بصفر.	
ceil ( )	double ceil (double x) ; يعود بـ x مقربة لأقرب رقم صحيح	<math. h>
clear ( )	void ios : : clear (int n=0) ; يغير حالة النهر إلى n	<iostream. h>
clearerr ( )	void clearerr (FILE* p) ; يصفر العلم end-of-file و error للملف P*	<stdio. h>
close ( )	void fstreambase : : close ( ); يقفل الملف الملحق للهدف المالك	<fstream. h>
cos ( )	double cos (double x); Returns the inverse cosine of x.	<math. h>
cosh ( )	double cosh (double x) ; Returns the hyperbolic cosine of x: $(e^x + e^{-x})/2$.	<math. h>
difftime ( )	double difftime (time_t t1, time_t t 0) يعود بالزمن بالثواني بين t0 و t1	<time. h>
eof ( )	int ios : : eof ( ) ; يعود بقيمة غير الصفرإذا كانت eofbit=1 ويعود بالصفر في غير ذلك .	<iostream. h>
exit ( )	void exit (int n) ; ينهي البرنامج ويعود بـ n للبرنامج المنادي	<stdlib. h>
exp ( )	double exp (double x) ; returns the exponential of z: e^x .	<math. h>
fabs ( )	double fabs (double x) ; Returns the absolute value of x.	<math. h>
fail ( )	int ios : : fail ( ) ; يعود بقيمة غير الصفر إذا كانت failbit=1 ويعود بصفر إذا كانت غير ذلك	<iostream. h>
fclose ( )	int fclose (FILE* p)	<stdio. h>

تابع الدوال سابقة التعريف

Function	Prototype and Description	Header File
	يقفل الملف P* ويفرغ كل العوازل. يعود بصفر إذا تمت هذه العملية بنجاح، ويعود بـ EOF في غير ذلك	
fgetc ( )	int fgetc (FILE* p) ; يقرأ ويعود بالحرف التالي من الملف P* وإلا فانه يعود بـ EOF	<stdio. h>
fgets ( )	char* fgets (char* s, int n, FILE* p); يقرأ السطر التالي من الملف P* ويخزنه في S* . الحرف الفارغ NULL يلحق بالحروف المخزنة في S . إذا لم تنجح العملية يعود بـ NULL	<stdio. h>
fill ( )	char ios : : fill ( ) يعود بحرف الحشو الحالي	<iostream. h>
	char ios : : fill (char c) ; يغير حرف الحشو الحالي إلى C ويعود بحرف الحشو السابق	
flags ( )	long ios : : flags ( ); يعود بأعلام التشكيل الحالية long ios : : flags (long n); يغير اعلام التشكيل الحالية إلى n ويعود بالاعلام السابقة	<iostream. h>
floor ( )	double floor (double x) ; يقرب x إلى أقرب أقل رقم صحيح	<math. h>
flush ( )	ostream& ostream : : flush ( ) ; يفرغ مخازن الخروج ويعود بالانهار الجديدة	<iostream. h>
fopen ( )	FILE* fopen (const char* p, const char* s) ; يفتح الملف P* ويعود بعنوان الهيكل الممثل للملف إذا نجح، ويعود بـ NULL في غير ذلك. السلسلة S تحدد حالة الملف: "r" للقراءة و "w" للكتابة ، و "a" للإلحاق ، و "r+" للقراءة والكتابة في ملف موجود ، و "w+" للقراءة والكتابة في ملف موجود ، و "a+" للقراءة والالحاق للف موجود	<stdio. h>
fprintf ( )	int fprintf (FILE* p, const char* s, . . .); تكتب خروج مشكلة للملف P* . تعود بعدد الاحرف التي كتبت اذا نجحت العملية، وإلا فإنها تعود بقيمة سالبة	<stdio. h>
fputc ( )	int fputc (int c, FILE* p); تكتب الحرف C في الملف P* . يعود بالحرف المكتوب وإلا فإنه يعود بـ EOF في حالة عدم النجاح	<stdio. h>

تابع الدوال سابقة التعريف

Function	Prototype and Description	Header File
fputs ( )	int fputs (const char* s, FILE* p); يكتب السلسلة S في الملف *P . يعود بقيمة غير سالبة إذا نجح وإلا فانه يعود بالقيمة EOF	<stdio. h>
fread ( )	size_t fread (void* x, size_t k, size_t n, FILE* p); يقرأ حتى عدد n عنصر له الحجم k من الملف *P ويخزنهم عند الموضع S في الذاكرة . يعود بعدد العناصر المقروءة	<stdio. h>
fscanf ( )	int fscanf (FILE* p, const char* s, ...) ; يقرأ دخل مشكل من الملف *P ويخزنهم عند الموضع S في الذاكرة. يعود بـ EOF إذا وصل نهاية الملف وإلا فإنه يعود بعدد العناصر المقروءة	<stdio. h>
fseek ( )	int fseek (FILE* p, long k, int base) ; Repositions the position marker of the file *p k bytes from its base, where base should be SEEK_SET for the beginning of the file, SEEK_CUR for the current position of the file marker, or SEEK_END for the end of the file. Returns 0 if successsful	<stdio. h>
ftell ( )	long ftell (FILE* p) ; يعود بموضع علامة مكان الملف *P وإلا فانه يعود بـ -1	<stdio. h>
fwrite ( )	size_t fwrite (void* s, size_t k, size_t n, FILE* p) ; يكتب n من العناصر كل منها له حجم k للملف *P ويعود بالرقم الذي تمت كتابته	<stdio. h>
gcount ( )	int istream : : gcount ( ) ; يعود بأخر الحروف التي تمت قراءتها	<iostream. h>
get ( )	int istream : : get ( ) ; istream& istream : : get (signed char& c); istream& istream : : get (unsigned char& c) ; istream& istream : : get (signed char* b, int n, char e = '\n') ; istream& istream : : get (unsigned char* b, int n, char e= '\n') ;	<iostream. h>

تابع الدوال سابقة التعريف

Function	Prototype and Description	Header File
	يقرأ الحرف التالي C من نهر الادخال . الاصدار الأول يعود بـ C أو EOF . الاصداران الأخيران يقران حتى n من الحروف في b ويقف عندما يجد e	
getc ()	int getc (FILE* p); Same as fgetc ( ) except implemented as a macro.	<stdio. h>
getchar ( )	int getchar ( ); يعود بالحرف التالي من مدخل قياسي وإلا فإنه يعود بـ EOF	<stdio. h>
gets ( )	char* gets (char* s); يقرأ الحرف التالي من مدخل قياسي ويخزنه في S . يعود بـ S أو NULL إذا لم يقرأ أي حرف	<stdio. h>
good ( )	int ios : : good ( ); يعود بقيمة غير الصفر اذا كانت حالة النهر صفر وإلا فإنه يعود بصفر	<iostream. h>
ignore ( )	istream& ignore (int n=1, int e=EOF); يستخلص حتى عدد n من الحروف من النهر ، أو حتى الحرف e ، ايهما يأتي أولاً . يعود بالنهر	<iostream. h>
isalnum ( )	int isalnum (int c); يعود بقيمة غير الصفر إذا كان e حرف هجاء أو رقم وإلا فإنه يعود بصفر	<ctype. h>
isalpha ( )	int isalpha (int c); يعود بقيمة غير الصفر إذا كان c حرف هجاء ، وإلا فإنه يعود بصفر	<ctype. h>
iscntrl ( )	int iscntrl (int c); يعود بقيمة غير الصفر إذا كان c حرف تحكم ، وإلا فإنه يعود بصفر	<ctype. h>
isdigit ( )	int isdigit (int c); يعود بقيمة غير الصفر إذا كان c رقم وإلا فإنه يعود بصفر	<ctype. h>
isgraph ( )	int isgraph (int c); يعود بقيمة غير الصفر إلى كان c حرف كتابة غير فاضي، وإلا فانه يعود بصفر	<ctype. h>
islower ( )	int islower (int c); يعود بقيمة غير الصفر إذا كان c حرف صغير ، وإلا فإنه يعود بصفر	<ctype. h>
isprint ( )	int isprint (int c); يعود بقيمة غير الصفر إذا كان c حرف قابل للطباعة ، وإلا فإنه يعود بصفر	<ctype. h>
ispunct ( )	int ispunct (int c); يعود بقيمة غير الصفر إذا كانت c حرف تشكيل وإلا فإنه يعود بصفر	<ctype. h>

تابع الدوال سابقة التعريف

Function	Prototype and Description	Header File
isspace ( )	int isspace (int c); يعود بقيمة غير الصفر إذا كان c أي حرف أبيض بما في ذلك الحروف ' ' و '\f' و '\n' و '\t' و '\v' وإلا فإنه يعود بصفر	<ctype. h>
isupper ( )	int isupper (int c) ; يعود بقيمة غير الصفر إذا كان c حرف كبير ، وإلا فإنه يعود بصفر	<ctype. h>
isxdigit ( )	int isxdigit (int c) ; يعود بقيمة غير الصفر إذا كان c واحد من العشرة أرقام أو واحد من الـ 12 رقم الستعشري ، وإلا فإنه يعود بصفر	<ctype. h>
labs ( )	long labs (long n) ; يعود بالقيمة المطلقة لـ n	<stdlib. h>
log ( )	double log (double x) ; Returns the natural logarithm (base e) of x.	<math. h>
log10 ( )	double log (double x) ; Returns the natural logarithm (base 10) of x.	<math. h>
memchr ( )	void* memchar (const void(s, int c, size_t k) ; يبحث العدد k من البايتات التي تبدأ عند S عن الحرف c . إذا وجد يعود بعنوان أول حدوث له ، وإلا فإنه يعود بـ NULL	<string. h>
memcmp ( )	int memcmp (const void* s1, const void* s2, size_t k); يقارن k بايت من الذاكرة تبدأ عند S1 مع k بايت من الذاكرة تبدأ عند S2 ويعود برقم سالب أو صفر أو موجب على حسب إذا كانت السلسلة الأولى أقل من أو تساوي أو أكبر من السلسلة الثانية	<string. h>
memcpy ( )	void* memcpy (const void* s1, const void* s2, size_t k) ; ينسخ الـ k بايت من الذاكرة والتي تبدأ عند S2 في مكان الذاكرة S1 ، ويعود بـ S1	<string. h>
memmove( )	int memmove (const void* s1, const void* s2, size_t k) ; مثل السابقة فيما عدا امكانية تداخل السلاسل	<string. h>
open ( )	void fstream : : open (const char* f, int m,	<fstream. h>

تابع الدوال سابقة التعريف

Function	Prototype and Description	Header File
	int p=filebuf : : openprot) ; void ifstream : : open (const char* f, int m=ios : : in, int p=filebuf : : openprot) ; void ofstream : : open (const char* f, int m=ios : : out, int p=filebuf : : openprot) ; يفتح الملف f في الحالة m والحماية p	
peek ( )	int istream : : peek ( ); يعود بالحرف التالي أو الـ EOF من نهر دون استخلاصه	<iostream. h>
pow ( )	double pow (double x, double y) ; Returns x raised to the power y (x^y) .	<math. h>
precision ( )	int ios : : precision ( ) ; int ios : : precision (int k) ; يعود بالدقة الحالية للنهر ، الصورة الثانية تغير الدقة الحالية إلى k وتعود بالدقة القديمة	<iostream. h>
tolower ( )	int tolower (int c) ; تعود بالصورة الصغيرة للحرف c إذا كان على الصورة الكبيرة ، وإلا فإنها تعود بالحرف c	<ctype. h>
toupper ( )	int toupper (int c) ; تعود بالصورة الكبيرة للحرف c إذا كان على الصورة الصغيرة ، وإلا فإنها تعود بالحرف c	<ctype. h>

الملحق G

الأرقام الستعشرية

نحن كبشر نستخدم عادة النظام العشرى decimal فى العد، ولقد سمى كذلك من الكلمة الاغريقية deka بمعنى (عشرة)، ولقد تعلم أجدادنا القدماء هذا النظام من العد على أصابعهم العشرة. إن الحاسبات لها إصبعان فقط (بمعنى أن كل بت يمكن أن تأخذ قيمتان فقط)، لذلك فإن النظام الثنائى يعمل جيداً مع الحاسبات. ولكن المشكلة مع النظام الثنائى أن تمثيل الأعداد فيه يتطلب سلسلة طويلة من البتات، فمثلاً الرقم 1996 يمثل كالتالى 11111001100 فى النظام الثنائى، ولذلك فإن البشر يجدون صعوبة فى التعامل مع هذه السلاسل الطويلة.

النظام الثنائى من السهل تحويله إلى أى نظام عد آخر إذا كانت قاعدة هذا النظام قوة من قوى العد 2. فمثلاً التحويل من النظام الثنائى إلى النظام الثمانى (القاعدة 8) يتطلب تجميع البتات الثنائية فى مجاميع كل منها من 3 بت، وبعد ذلك نترجم كل مجموعة إلى ما يناظرها فى النظام الثمانى. فمثلاً تحويل الرقم 11111001100 إلى النظام الثمانى يكون كالتالى :

11,111,001,100 = 3714

حيث 11 تحولت إلى 3 ، و 111 تحولت إلى 7، و001 تحول إلى 1، و 100 تحولت إلى 4. التحويل من النظام الثمانى للثنائى سهل جداً فمثلاً الرقم 2650 يتحول إلى 010110101000 والذى يقابل 1448 فى النظام العشرى. لاحظ أن الأعداد الثمانية هى فقط الأعداد من 0 إلى 7 أى 7, 6, 5, 4, 3, 2, 1, 0.

بعد الرقم 8 فإن العدد التالى الذى هو قوة من قوى الرقم 2 هو العدد 16. إن استخدام نظام العد الستعشرى يجعل الأعداد تمثل فى عدد أقل من الخانات. ولقد اشتق الأسم hexadecimal من الكلمة الإغريقية hex بمعنى ستة و deka بمعنى عشرة. التحويل من النظام الثنائى إلى الستعشرى سهل تماماً مثل التحويل من الثنائى إلى الثمانى، فمثلاً، الرقم 10111010100 يتم تحويله إلى النظام الستعشرى عن طريق وضعه فى مجاميع كل منها من 4 بت (من اليمين لليسار)، وبعد ذلك تترجم كل مجموعة إلى ما يناظرها فى النظام الستعشرى. لذلك فإن الرقم السابق يتم تحويله كالتالى :

0101,1101,0100 = 5d4

حيث 0101 تقابل 5، و 1101 تقابل d، و 0100 تقابل 4 فى النظام الستعشرى. الأرقام 10، 11، 12، 13، 14، 15 يرمز لها بالحروف الستة الهجائية a، b، c، d، e، f. مناولات الخرج dec، و hex ، و oct تستخدم للتحويل بين النظم المختلفة.

مثال G. 1 هذا المثال يبين كيف نطبع عنوان وقيمة متغير معين:

```
# include <iostream. h>
main ( )
{
    int n = 1492 ;    //  base 10
    cout << "Base  8 : n  =  " << oct << n << endl ;
    cout << "Base 10 : n  =  " << n  << n << endl ;
    cout << "Base 16 : n  =  " << hex << n << endl ;
}
```

```
Base  8 : n = 2724
Base 10 : n = 1492
Base 16 : n = 5d4
```

فى هذا المثال تم استخدام المناول oct لتحويل الخرج التالى إلى النظام الثمانى. لاحظ أن الخرج يعود ثانية إلى النظام العشرى إلى أن يتم استخدام مناول النظام الستعشرى hex.

المثال التالى يبين كيفية إدخال أرقاماً صحيحة فى النظام الثمانى والستعشرى. الأعداد الثمانية يلحق بها الحرف o والأرقام الستعشرية يلحق بها الحرفين ox كدلالة لهذه الأنظمة.

مثال G2 يوضح هذا المثال كيفية طبع القيمة والعنوان لمتغير

```
# include <iostream. h>
main ( )
{
    int n ;
    cout << "Enter an octal numeral (use 0 prefix) : " ;
    cin >> n ;
    cout << "Base  8 : n  =  " << oct << n << endl ;
    cout << "Base 10 : n  =  " << dec << n << endl ;
    cout << "Base 16 : n  =  " << hex << n << endl ;
    cout << "Enter a decimal numeral : " ;
    cin >> n ;
    cout << "Base  8 : n  =  " << oct << n << endl ;
    cout << "Base 10 : n  =  " << dec << n << endl ;
    cout << "Base 16 : n  =  " << hex << n << endl ;
    cout << "Enter a hexadecimal numeral (use ox prefix) : " ;
```

```
    cin >> n ;
    cout << "Base  8 : n  =  " << oct << n << endl ;
    cout << "Base 10 : n  =  " << dec << n << endl ;
    cout << "Base 16 : n  =  " << hex << n << endl ;
}
```

```
Enter an octal numeral (use o prefix) : 0777
cout << "Base  8 : 777
cout << "Base 10 : 511
cout << "Base 16 : 1ff
Enter an octal numeral : 511
cout << "Base  8 : 777
cout << "Base 10 : 511
cout << "Base 16 : 1ff
Enter a hexadecimal numeral (use ox prefix) : ox1ff
cout << "Base  8 : 777
cout << "Base 10 : 511
cout << "Base 16 : 1ff
```

PROGRAMMING WITH C++
(Schaum)



OVER 30 MILLION SOLD

تعريف بسلسلة شـــوم : لمـــاذا تشترى كتاب شــوم؟

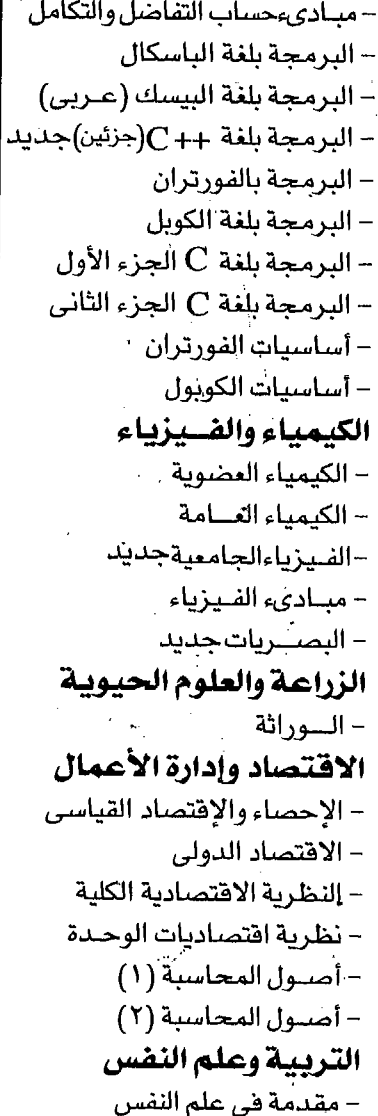
كل كتاب يحتوى على النظرية الأساسية والتعريفات ومئات من المسائل المحلولة بعناية وكذلك ...مسائل غير محلولة لمساعدة الطالب على التفوق.

الهندســة

- المبادىء الرقمية
- تكنولوجيا الإلكترونيات
- الدوائر الكهربائية جديد
- الماكينات الكهربية
- نظم القوى الكهربية
- النبائط الإلكترونية ودوائرها
- أساسيات الهندسة الكهربائية جديد
- الديناميكا الحرارية
- مقاومة المـــواد
- ميكانيكا الموائع والهيدروليكا
- اهتزازات ميكانيكية
- الميكانيكا الهندسية - استاتيكا
- الميكانيكا الهندسية - ديناميكا

الرياضيات والحاسبات

- الاحـتمالات
- الإحصـــاء
- بحوث العمليات
- التحليل العددى
- تحليل المتجهات
- الجـبر الخـطـي
- التفاضل والتكامل المتقدم
- حساب التفاضل والتكامل
- الدوال المركـبة
- الرياضيات الأساسية للحاسب
- الرياضيات المتقدمة
- المعادلات التفاضلية جديد
- الميكانيكا العـــامة
- نظـرية الفـئة



- مبادىء حساب التفاضل والتكامل
- البرمجة بلغة الباسكال
- البرمجة بلغة البيسك (عـربى)
- البرمجة بلغة ++C (جزئين) جديد
- البرمجة بالفورتران
- البرمجة بلغة الكوبل
- البرمجة بلغة C الجزء الأول
- البرمجة بلغة C الجزء الثانى
- أساسيات الفورتران
- أساسيات الكوبول

الكيمياء والفـيزياء

- الكيمياء العضوية
- الكيمياء العـــامة
- الفيزياء الجامعية جديد
- مبـادىء الفـيزياء
- البصـريات جديد

الزراعة والعلوم الحيوية

- الـــوراثة

الاقتصاد وإدارة الأعمال

- الإحصاء والإقتصاد القياسى
- الاقتصاد الدولى
- النظرية الاقتصادية الكلية
- نظرية اقتصاديات الوحـدة
- أصـول المحاسبة (١)
- أصـول المحاسبة (٢)

التربية وعلم النفس

- مقدمة فى علم النفس
- سيكولوجية التعـلم



INTERNATIONAL HOUSE FOR CULTURAL INVESTMENTS

P.O.Box 5599 Heliopolis West. Cairo/Egypt
Tel.: 2972344 - 2957655, Fax:(00202) 2957655